Model-based Monitoring and Diagnosis of Systems with Software-Extended Behavior

by

Tsoline Mikaelian

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Master of Science in Aeronautics and Astronautics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2005

Author ....
Department of Aeronautics and Astronautics
May 20, 2005

Certified by....................................
Brian C. Williams
Associate Professor of Aeronautics and Astronautics
Thesis Supervisor

Accepted by....................................
Jaime Peraire
Professor of Aeronautics and Astronautics
Chair, Committee on Graduate Students

Model-based Monitoring and Diagnosis of Systems with Software-Extended Behavior

by

Tsoline Mikaelian

Submitted to the Department of Aeronautics and Astronautics
on May 20, 2005, in partial fulfillment of the
requirements for the degree of
Master of Science in Aeronautics and Astronautics

Abstract

Model-based diagnosis of devices has largely operated on hardware systems. However, in most complex systems today, such as aerospace vehicles, automobiles and medical devices, hardware is augmented with software functions that influence the system's behavior. As these sophisticated systems are required to perform increasingly ambitious tasks, there is a growing need to ensure their robustness and safety.

Prior work introduced probabilistic, hierarchical, constraint automata (PHCA), to allow compact encoding of both hardware and software behavior. The contribution of this thesis is a capability for monitoring and diagnosing software-extended systems in the presence of delayed symptoms, based on the expressive PHCA modeling formalism. Hardware models are extended to include the behavior of associated embedded software, resulting in more comprehensive diagnoses.

This work introduces a novel approach that frames diagnosis over a finite time horizon as a soft constraint optimization problem (COP), which is then decomposed into independent subproblems using tree decomposition techniques. There are two advantages to this approach. First, the approach enables finite-horizon diagnosis in the presence of delayed symptoms. Second, the soft COP formulation provides convenient expressivity for encoding the PHCA models and their execution semantics, and enables the use of decomposition-based, efficient optimal constraint solvers. The solutions to the COP correspond to the most likely state trajectories of the software-extended system. These state trajectories are enumerated and tracked within the finite receding horizon, as observations and issued commands become available.

The diagnostic capability has been implemented and demonstrated on several scenarios from the aerospace and robotic domains, including vision-based rover navigation, the global metrology subsystem of the MIT SPHERES satellites, and models of the NASA New Millennium Earth Observing One (EO-1) spacecraft.

Thesis Supervisor: Brian C. Williams
Title: Associate Professor of Aeronautics and Astronautics

Acknowledgments

First and foremost, I would like to thank my family for their love and support. I am eternally grateful to my parents, Alice and Hratch Mikaelian, and my uncle Zareh Mikaelian, for their devotion and sacrifices that paved my way to great opportunities. I would like to thank my sister Shoghig for her friendship and moral support.

I would like to especially thank my supervisor, Professor Brian C. Williams, for providing me with this research opportunity. I greatly appreciate his technical guidance, ideas and insights that made this thesis possible.

I would like to thank Jonathan Babb for his moral support and insights through writing drafts of my thesis and related papers. His innovative thinking, entrepreneurial spirit and superior leadership have inspired me tremendously.

I would like to thank all my colleagues at the Model-based Embedded and Robotic Systems (MERS) group, the Space Systems Lab (SSL), and the Computer Science and Artificial Intelligence Lab (CSAIL), for invaluable discussions and suggestions related to this work. Thanks to Martin Sachenbacher for providing an implementation of the constraint solver used in this work.

I would like to thank the Zonta International Foundation for support and encouragement provided through the Amelia Earhart Fellowship Award.

This research has been sponsored in part by the DARPA NEST program under contract F33615-01-C-1896 and by the DARPA SRS program under contract FA8750-04-2-0243.

The EO-1 models, used in validating this work, were provided by the NASA Ames Research Center.

Contents

1 Introduction **13**

1.1 Robustness of Complex Systems 13

1.2 Vision-based Navigation Scenario 15

1.3 Diagnosing Software-Extended Systems 16

1.3.1 Challenges 16

1.3.2 Related Work and Innovative Claims 17

1.3.3 Diagnostic System Architecture 22

1.4 Thesis Contributions 25

1.5 Outline 26

2 Modeling and Monitoring Software-Extended Systems **29**

2.1 Modeling Software-Extended Behavior 29

2.2 Probabilistic, Hierarchical Constraint Automata 33

2.3 Best-First Trajectory Enumeration for PHCA 36

2.4 N-Stage Best-First Trajectory Enumeration 38

2.5 The N-BFTE Process 40

2.5.1 Offline Phase 41

2.5.2 Online Phase 41

2.6 Summary 42

3 Offline Phase: Formulation of COP **45**

3.1 Encoding the N-BFTE Problem for PHCA 45

3.1.1 Soft and Hard Constraints 46

3.1.2 Elements of the COP Formulation 49
3.1.3 Encoding the PHCA Execution Semantics 51
3.1.4 Consistency Constraints 52
3.1.5 T=0 Constraints 55
3.1.6 Transition Constraints 59
3.1.7 Marking Constraints 69
3.1.8 Special Cases of Marking Constraints 74
3.1.9 Summary of Constraint Encoding 78
3.2 Tree Decomposition of the COP 79
3.3 Summary of the Offline Phase 81

4 Online Phase: Best-First Trajectory Tracking 83
4.1 PHCA State Trajectories 83
4.2 The N-BFTE Algorithm 84
4.3 Parameter Selection 89
4.4 Summary 89

5 Results and Discussion 91
5.1 Implementation 91
5.2 Demonstration Scenarios 92
5.2.1 SPHERES Global Metrology 93
5.2.2 Earth Observing One 94
5.3 Results 94
5.4 Future Work 98
5.5 Conclusions 99

A List of PHCA Encoding Constraints 101

B Probabilistic Concurrent Constraint Automata 105

List of Figures

1-1 Vision-based navigation of MERS Group rovers. 15

1-2 Process diagram for PHCA-based diagnosis 23

2-1 Camera module for navigation system 30

2-2 Behavioral model (PCCA) for the camera component. 30

2-3 Most likely diagnoses of the camera module based on hardware component models. Nominal state = no failures. 31

2-4 Most likely diagnoses of the camera module based on the software-extended behavior models. 32

2-5 PHCA model for the camera/image processing module. Circles represent primitive locations, boxes represent composite locations and small arrows represent start locations. 34

2-6 Missed diagnosis as a result of tracking a limited number of trajectories (K-Best) 37

2-7 (a) Trellis diagram showing the evolution of an approximate belief state; (b) branching tree representation showing state trajectories. . . 40

3-1 PHCA models for spacecraft subsystems. 56

3-2 PHCA example with initial start probabilities. 59

3-3 Transitions disallowed from composite PHCA locations. 60

3-4 PHCA transitions with multiple targets. 61

3-5 Semantics of transition choice, target marking, and composite full marking. 62

3-6 PHCA with two probabilistic transitions. 63

3-7 Nested start locations; each start location is indicated by a small arrow. 66

3-8 Target Identification example. P is a primitive start location; T1, T2 and T3 are transitions; start(P) indicates that P is a start location. . 67

3-9 Marking example scenario. 70

3-10 Categories of PHCA locations: (a) direct target and start location; (b) direct target and non-start location; (c) indirect target and start location; (d) indirect target and non-start location. Category (d) is possible only for composite locations. P refers to a primitive location; C refers to a composite location; T1 refers to a transition; start refers to a start location. 75

3-11 Hypergraph of a constraint network. 79

3-12 A tree decomposition of the hypergraph in Figure 3-11. 81

4-1 Online N-BFTE process. 85

4-2 Addition of a unary constraint to the tree decomposition in Figure 3-12. 86

4-3 N-BFTE example. Each node represents a PHCA state (marking). A trajectory is represented as a sequence of filled nodes. The dotted lines represent a single-step shift of the time horizon. 87

5-1 The SPHERES testbed. [32, 45] 93

5-2 Size of the COP generated offline for the SPHERES and EO-1 models. 95

5-3 Constraint graph of the EO-1 model for $N = 1$. 96

5-4 Tree decomposition of the EO-1 constraint graph in Figure 5-3 96

5-5 Online performance for the SPHERES and EO-1 models. 97

List of Tables

3.1 State consistency for camera "Off" location, for generic time t. 54

3.2 Guard consistency constraint for $\tau \equiv (Initializing \rightarrow TakingPicture)$, for generic time t. 54

3.3 Model Marking at T=0 56

3.4 T=0 Full Marking constraint for the camera in Figure 2-5. 57

3.5 T=0 Probabilistic Marking example. The * notation indicates the full domain of the variable; in this case {*Enabled, Disabled*} 59

3.6 Probabilistic transition choice constraint. 63

3.7 Transition constraint for the $(Initializing \rightarrow TakingPicture)$ transition, represented by variable τ at time t. Each * represents the full domain of the variable in the corresponding column. 64

3.8 Target identification constraint. Each * represents the full domain of the variable in the corresponding column. 67

3.9 Full marking of $C1$ and $C2$ from Figure 3-7. Each * represents the full domain {*Enabled, Disabled*}. 68

3.10 Marking of primitive location *P1* in Figure 3-9. 71

3.11 Composite marking of the $C1$ and $C2$ locations in Figure 3-9. Each * represents the full domain of the variable in the corresponding column. 72

3.12 Hierarchical composite marking/unmarking of the $C1$ and $C2$ locations in Figure 3-9. $Subautomata(C1) = \{P1,P2,C2\}$ and $Subautomata(C2) = \{P3,P4,P5\}$. Each * represents the full domain of the variable in the corresponding column. 73

5.1 Number of constraints and variables generated for the camera model in Figure 2-5. N is the size of the time horizon. 93

Chapter 1

Introduction

1.1 Robustness of Complex Systems

Many complex systems today, such as aerospace vehicles, robotic networks, automobiles and medical devices, consist of a mix of hardware and software components that may malfunction or interact in unexpected ways, potentially leading to failures. As these sophisticated systems are required to perform increasingly ambitious tasks, there is a growing need to ensure their robustness and safety. For instance, the Jupiter Icy Moons Orbiter (JIMO) and the Mars Science Laboratory (MSL) missions are expected to handle long term operations in harsh and uncertain space and planetary environments, while executing complex mission tasks to achieve scientific objectives.

Ensuring the robustness and safety of complex systems is a top priority and major challenge, given recent catastrophic events, such as the Space Shuttle Columbia accident and the loss of the Mars Polar Lander (MPL). In addition to uncovering flaws in the safety culture during system development and testing, the Columbia accident demonstrates the catastrophic consequence of failing to detect and respond to unexpected events in the Shuttle's operational environment. In the case of MPL, the leading hypothesis for its loss is an anomaly during its entry, descent and landing sequence: the MPL software prematurely shut down the engine upon detection of spurious signals from a faulty touchdown sensor, leading to the loss of the lander. The root of the problem was that a sensor failure was not detected, and even if it was detected, the software wouldn't protect against a failed sensor. Yet another example demonstrating disastrous behavior is the Therac-25 [35], a medical device for radiation therapy. The Therac-25 overdosed six people because of software errors that allowed the device to operate in a hazardous mode. A lesson to be learned from such catastrophic accidents is that complex systems are prone to behave in an unexpected fashion under some conditions. These conditions may have been overlooked during system design and development.

The need for robustness of increasingly ambitious applications can be addressed through an intelligent onboard fault management system for monitoring complex behavior and diagnosing faults. Model-based systems perform fault management by automated reasoning over declarative models of system behavior [53]. Model-based diagnosis of devices has traditionally operated on hardware models [26, 15, 21]. For instance, given an observation sequence, the Livingstone [54] diagnostic engine estimates the state of hardware components based on hidden Markov models that describe each component's behavior in terms of nominal and faulty modes. At the other end of the spectrum, researchers have applied model-based diagnosis to software debugging [9, 39, 40].

This work explores the middle ground between model-based hardware diagnosis and software debugging, resulting in a novel capability for monitoring and diagnosing systems with combined hardware and software behavior. This capability is crucial, given that the functionality of many hardware components is extended or controlled by embedded software. Two examples of devices with software-extended behavior are a communications module with an associated device driver, and an inertial navigation unit with embedded software for trajectory determination. The embedded software in each of these systems interacts with the hardware components and influences their behavior. In order to correctly estimate the state of these devices, it is essential to consider their software-extended behavior, as it may provide additional evidence that substantially alters the system's diagnoses. The following sections further motivate this work through the vision-based navigation scenario, and present the major challenges, proposed approach and contributions of this thesis.

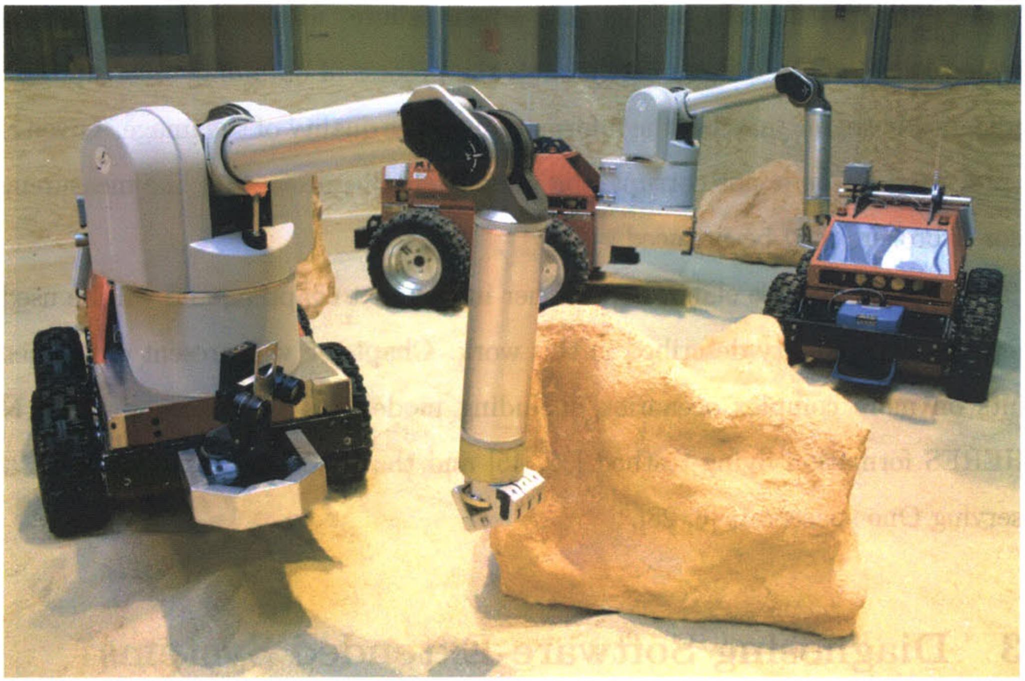

Figure 1-1: Vision-based navigation of MERS Group rovers.

1.2 Vision-based Navigation Scenario

As an example of a complex system, consider vision-based navigation for an autonomous rover exploring the surface of a planet (Figure 1-1). The camera used within the navigation system is an instance of a device that has software-extended behavior: the image processing software embedded within the camera module augments the functionality of the camera by processing each image and determining whether it is corrupt. A sensor measuring the camera voltage may be used for estimating the physical state of the camera. A hardware model of the camera describes its physical behavior in terms of inputs, outputs and available sensor measurements.

A diagnosis engine, such as Livingstone [54], that uses only hardware models will not be able to reason about a corrupt image. The embedded software provides additional information on the quality of the image that is essential for correctly diagnosing the navigation system. To see why this is the case, consider a scenario in which the camera sensor measures a zero voltage. Based solely on hardware models of the camera, the measurement sensor and the battery, the most likely diagnoses will include camera failure, low battery voltage and sensor failure. However, given a software-extended model of the camera that models the process of obtaining a corrupt image, the diagnostic engine may use the information on the quality of the image. Given that the processed image is not corrupt, the most likely diagnosis, that the measurement sensor is broken, may be deduced.

This scenario will be elaborated further in the next chapters, and will be used to illustrate the capability described in this work. Chapter 5 will present experimental results on more complex scenarios, including models of two spacecraft: the MIT SPHERES formation flying testbed [32, 45] and the NASA New Millennium Earth Observing One spacecraft [6, 28].

1.3 Diagnosing Software-Extended Systems

This section highlights the major challenges in monitoring and diagnosing software-extended systems, presents an overview of prior work, and emphasizes the novel approach and contributions of this thesis.

1.3.1 Challenges

The vision scenario introduced in the previous section demonstrates that a diagnostic engine for complex systems with software-extended behavior must:

1. Monitor the behavior of both the hardware and embedded software, so that the software state can be used for diagnosing the hardware. This requires an expressive modeling framework for capturing complex software and hardware behaviors.

2. Reason about the system state, given delayed symptoms. An instance of a delayed symptom is the image quality determined by the camera software after it has completed all stages of image processing.

3. Efficiently compute the leading estimates of system state.

1.3.2 Related Work and Innovative Claims

This section addresses the differences between previous work and the approach introduced in this thesis for tackling the three challenges highlighted in the previous section. First, we compare this work to research on model-based software debugging, which is focused on finding software bugs and not on diagnosing combined software and hardware systems. Second, we discuss previous work on modeling and diagnosing complex systems, with emphasis on building upon that work and the innovative claims relative to that work. Third, we discuss previous work on model-based diagnosis in order to address the challenge of delayed symptoms, and motivate the generalization of this capability to software-extended systems. Fourth, we discuss research on the core computational approaches that are relevant to performing diagnosis efficiently. Finally, we discuss model simulation used in previous diagnostic systems, along with the differences from this work.

Model-based Software Debugging

Much research has focused on modeling software and the application of model-based diagnosis to software debugging [40, 39, 9, 25]. Model-based debugging was first introduced in [9] for logic based declarative languages. This work was later extended to imperative and object-oriented languages [39, 40] and to component-based software [25]. The key point here is that these approaches do not deal with the diagnosis of combined hardware and software systems.

In contrast to previous work on model-based software debugging, this work leverages information within the embedded software to refine the diagnoses of physical systems. As such, the problem of diagnosing software bugs is not addressed in this work. Without loss of generality, it is assumed that software bugs discovered at runtime are handled by a separate exception handling mechanism. However, given software fault models, the work in this thesis can be applied to automated verification of embedded software.

Modeling and Diagnosing Complex Systems

Challenge 1 presented in the previous section highlights the need for an expressive modeling framework to specify complex behaviors, in particular that of embedded software. Models typically used by previous model-based diagnostic engines such as GDE [15], Livingstone [54, 33] and Titan [38], do not support constructs for specifying complex behaviors, such as hierarchical structure and multiple transitions enabled within a single automaton. This is a major limitation for modeling complex systems, and software-extended systems in particular. Capturing the behavior of software is much more complex than that of hardware, due to the hierarchical structure of a program and the use of complex constructs such as iteration, preemption, sequential and concurrent execution. This complexity was addressed in prior work [52], by introducing Probabilistic, Hierarchical, Constraint Automata (PHCA) to uniformly and compactly encode both software and hardware behavior. Hierarchical automata have been introduced for reactive embedded languages such as Esterel [4] and State Charts [27]. However, PHCA offer additional features [52] that are key for model-based diagnosis of complex systems. These features include traversing multiple transitions simultaneously and enabling transitions conditioned on what can be deduced, and not just what is explicitly assigned. In addition to introducing PHCA, the work in [52] also has introduced the idea of PHCA-based state estimation as K-best enumeration, which involves stepping the PHCA automata for each of its K most likely transitions, checking consistency with observations and commands, and performing belief state update at each time step.

Building upon the PHCA modeling framework described above, this work introduces a novel approach to PHCA-based monitoring and diagnosis. This approach is implemented and validated on a variety of scenarios to be discussed in the next chapters. The novelty of the approach in this thesis is threefold. First, it enables PHCA-based reasoning over several time steps to account for delayed symptoms encountered during diagnosis. This is accomplished by framing diagnosis as the task of N-Stage Best-First Trajectory Enumeration (N-BFTE), where N is the number of time steps within the horizon. Second, it relies on constraint decomposition to enhance the efficiency of the diagnosis algorithm. Third, it integrates the model simulation (stepping of the automata) and consistency checking (with observations and commands) into a single constraint optimization framework. Each of these innovations for handling delayed symptoms, efficiency and model simulation, are discussed further in the following three subsections.

Delayed Symptoms

The challenge of diagnosing hardware systems in the presence of delayed symptoms was addressed by the Livingstone-2 (L2) [33] diagnostic engine. The Livingstone-1 [54] and Titan [53] engines enumerate and maintain only the most likely system state trajectories. This corresponds to maintaining an approximate *belief state* - a probability distribution over the possible system states - at any point in time. However, this approximation leads to incorrect diagnoses when an unmaintained trajectory becomes very likely given delayed evidence. L2 was introduced to address the problem of delayed symptoms, by incrementally generating the approximate belief state; this means that if the maintained trajectories are consistent with observations and commands, only a partial belief state is maintained, similar to Livingstone and Titan. On the other hand, if inconsistencies arise, L2 backtracks to revisit past trajectories, and generates a new belief state. In order to accomplish this, L2 maintains a history of observations and commands over an infinite time horizon; however, to achieve tractability, L2 uses approximate system models over the recent past, and summarization over the distant past. Furthermore, L2 resorts to a ranking system [33], which is an order of magnitude approximation to model probabilities. The likelihood of a trajectory in L2 is defined in terms of ranks. Finally, L2 uses conflict coverage search to enumerate and track the most likely trajectories.

This work presents a capability for handling delayed symptoms by posing the PHCA-based diagnosis problem over a finite time horizon. Thus, given a history of observations and commands within a receding N-Stage time horizon, the most likely trajectories of system state that are consistent with the observations, commands and the PHCA models, are enumerated and tracked. This approach has several key differences from L2. First, PHCA models are used as the basis of the diagnostic system presented in this work. As discussed above, PHCA models offer richer expressivity than L2 models for specifying complex behaviors, such as embedded software. Second, diagnosis is limited to a finite time horizon, in order to limit model growth. This is in contrast to using approximations to achieve tractability. However, since both of these approaches are approximations to considering an infinite horizon, they are expected to be roughly equivalent in terms of handling delayed symptoms. Third, a limited number of state trajectories are tracked at all times, in the spirit of Livingstone-1 and Titan. The benefit of this is to avoid a large increase in the number of trajectories tracked by L2 when inconsistencies arise due to delayed symptoms. Finally, this work relies on decomposition-based constraint solvers to achieve efficiency, while L2 uses conflict-directed search. The use of conflicts has proven very effective in diagnostic systems. On the other hand, recent advances in decomposition techniques have motivated its use in this work. The following section sheds more light on the major paradigms for addressing the efficiency challenge.

Computational Efficiency

Challenge 3, as highlighted in the previous section, is to efficiently compute the leading diagnoses of a software-extended system. Prior work on model-based diagnosis has addressed this challenge in the following prominent ways. First, the number of state trajectories maintained are typically limited to the K most likely ones [13, 54]. This approximation efficiently tackles the intractability of maintaining all possible state trajectories that are exponential in the number of components in the system. Second, diagnostic engines have made extensive use of conflicts and their derivatives, such as kernel diagnoses, to efficiently compute the leading diagnoses [14, 55, 54, 16, 20, 38, 53]. For example, the OPSAT optimal satisfaction engine implements the conflict-directed A* search algorithm presented in [55], which uses conflicts to jump over diagnosis candidates that do not resolve the conflicts. Third, model compilation has been used to achieve real-time efficiency, such as in the MiniMe engine [7]. Fourth, structure-based approaches to diagnosis were proposed to address the efficiency challenge by exploiting system structure [10, 11, 18, 19, 1]. While some approaches [10, 11] focus on a symbolic logic formulation for exploiting system structure, others [18, 19] propose exploiting the structure of constraints among multivalued variables.

This work draws upon the benefits of three of the above approaches: limiting the number of trajectories tracked, offline model compilation, and structural decomposition. The number of state trajectories tracked are limited in order to achieve tractability, similar to previous engines such as Livingstone [54] and Titan [53]. However, as opposed to previous work that uses conflict-directed search, this work advocates the use of tree decomposition of constraints for model-based diagnosis, in the spirit of [18, 19]. This is motivated by recent advances in tree decomposition techniques [24, 17, 31] and decomposition-based optimal constraint solvers [47, 48]. The diagnostic problem is thus formulated and decomposed in an offline phase, and solved online as observations and commands become available.

Model Simulation

Model-based diagnosis algorithms typically apply a simulation step to predict model behavior, followed by a consistency checking step to refute diagnosis candidates that are inconsistent with observations and commands [53, 38].

The approach introduced in this thesis differs from previous approaches in that it combines the simulation and consistency phases by encoding the PHCA automata and their execution semantics as constraints. This encoding is particularly convenient for reasoning over several time steps, as the simulation and consistency checking steps are reduced into the single task of solving a constraint optimization problem (COP) over the finite horizon. More specifically, the automata and their execution semantics are mapped to a soft COP [50, 47, 2] that supports associating probabilities with arbitrary constraints, rather than just decision (solution) variables. For instance, the soft constraint framework allows associating probabilities with constraints that encode PHCA transitions, while solving for PHCA state variables, as described further in Chapter 3. This enables encoding uncertainty in the constraints, in order to simulate the PHCA models.

The following section presents the diagnostic system architecture that incorporates the innovative claims discussed above. The next chapters elaborate the design, development and demonstration of the diagnostic system.

1.3.3 Diagnostic System Architecture

This work introduces a novel monitoring and diagnostic system for software-extended systems, based upon several key innovations, as identified in the previous section. The main features of this capability are as follows. First, the diagnostic system is based on the expressive PHCA modeling formalism [52] for capturing complex behaviors. PHCA-based diagnosis is defined as the task of enumerating and tracking the K most likely PHCA state trajectories. Second, the diagnostic task is posed over a finite time horizon, in order to account for delayed symptoms. Third, diagnosis is framed as a soft constraint optimization problem (COP) [50] that encodes the PHCA models and their execution semantics over the finite horizon. This formulation integrates the PHCA simulation phase into the COP, thereby reducing the diagnosis task to that of solving the COP and tracking the most likely solutions, which correspond to the most likely PHCA state trajectories. Fourth, tree decomposition [31, 17, 24] is applied to the COP to decompose it into independent subproblems by exploiting the structure of the constraints, thereby enabling the use of efficient optimal constraint solvers [47, 48].

The PHCA-based diagnostic system architecture consists of two phases: an offline compilation phase and an online solution phase, as shown in Figure 1-2. The offline compilation phase precedes the online monitoring and diagnosis phase, and does not require the availability of any observations or issued commands. The following describe the key features of each phase.

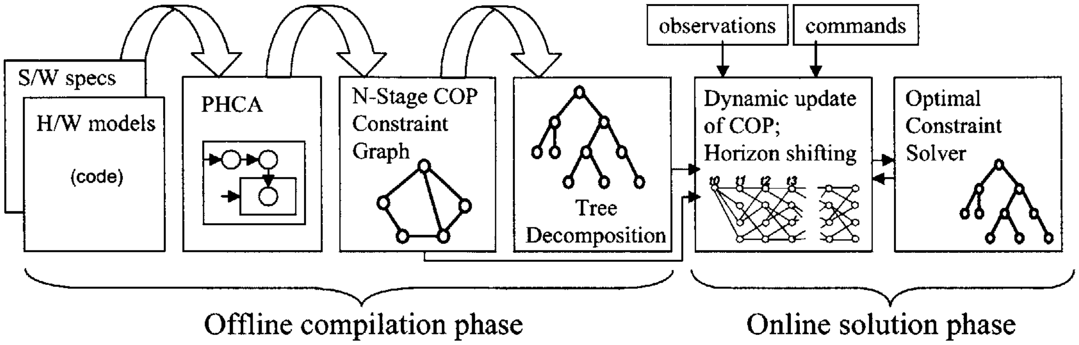

Figure 1-2: Process diagram for PHCA-based diagnosis

Offline Compilation:

As illustrated in Figure 1-2, the offline compilation phase consists of three stages that specify the PHCA models, formulate diagnosis as a COP over a finite (N-Stage) time horizon, and apply tree decomposition to the COP. The offline phase sets the infrastructure for the online monitoring and diagnosis phase, and enables the efficiency of the online process. As mentioned above, observations and issued commands are not required for the offline phase; those are incorporated into the online diagnostic process described in the next section.

In the offline phase, specifications of complex, software-extended behavior are compiled to PHCA models. These specifications are coded in the high-level Reactive Model-based Programming Language (RMPL) [52]. The automatic compilation of RMPL specifications to PHCA is presented in [52], and is outside the scope of this work. PHCA models can be represented graphically, as illustrated in the following chapters.

Given PHCA models, diagnosis is defined as the task of enumerating and tracking the most likely PHCA state trajectories within a finite, N-Stage time horizon. The finite-horizon diagnosis accounts for delayed symptoms. The PHCA state trajectory estimation task is formulated as a soft constraint optimization problem (COP) [50] within the N-Stage horizon. The COP encodes the PHCA models and their execution semantics as probabilistic constraints, such that the optimal solutions corresponds to the most likely PHCA state trajectories. Encoding the PHCA execution semantics as constraints eliminates the need for a separate model simulation step during the online phase. Furthermore, soft constraints provide convenient expressivity for encoding the models, by not limiting uncertainty specification to decision variables [50].

Finally, tree decomposition [24, 31] is applied to the constraint network to exploit the independence of subproblems through local consistency and dynamic programming, thus enabling the use of efficient optimal constraint solvers [47, 48] during the online phase.

For example, consider the vision-based navigation scenario introduced in Section 1.2. In the offline phase, specifications of the image processing software, as well as the physical behavior of the hardware components, such as the camera, can be captured as PHCA. The PHCA is then compiled into a soft COP that encodes the models over several time steps. This enables accounting for delayed evidence, such as the image quality, during the online diagnosis phase. The offline compilation does not require any observations (such as the image quality) or issued commands (such as turning on the camera). Furthermore, the offline COP formulation enables the online simulation of the PHCA models, and thereby the behavior of the camera module, through constraints that encode the models and their execution semantics. The encoding of PHCA models as soft COP posed over a finite horizon, and the decomposition of the resulting COP are presented in detail in Chapter 3.

Online Trajectory Tracking:

The online phase uses the offline COP formulation and its tree decomposition, to enumerate and track the most likely PHCA state trajectories that are consistent with observations and commands within the shifting N-Stage horizon. Recall that the COP generated offline does not specify assignments to observations and commands; those are available in the online phase. Therefore, the COP must be updated dynamically in the online phase.

The COP is updated in the online phase by shifting the time horizon, incorporating new observations and commands, and inserting constraints for tracking the trajectories found within the previous horizon. Within each time horizon, the COP is solved using an efficient, decomposition-based optimal constraint solver [47]. Similar to previous diagnostic engines [54, 53, 38], the solutions to the COP are enumerated in best-first order, thus efficiently focusing on the most likely trajectories while allowing anytime behavior.

For the vision-based navigation example, observations such as the image quality are added to the COP in the online phase. The COP is then solved to generate the most likely diagnoses that are consistent with the observations and commands over the N-Stage horizon. As the time horizon is shifted, the trajectories computed within the previous horizon must be tracked. This is accomplished by adding trajectory tracking constraints to the COP, which is described further in Chapter 4.

1.4 Thesis Contributions

The contributions of this work are as follows.

- First, a novel capability is introduced for diagnosing combined hardware and software systems in the presence of delayed symptoms, based on an expressive modeling framework called Probabilistic Hierarchical Constraint Automata (PHCA) [52]. This is accomplished by framing PHCA-based diagnosis as a constraint optimization problem (COP) based on soft constraints [50] that encode the structure and semantics of PHCA. This formulation is particularly appropriate for encoding complex models since probabilities may be associated with arbitrary sets of variables rather than only decision variables. Furthermore, encoding the execution semantics as constraints integrates consistency checking with model simulation over the time horizon. This integration eliminates the need for interleaving separate model simulation and consistency checking steps over several time steps.

- The second contribution is the ability of the PHCA-based diagnostic system to reason about faults in the presence of delayed symptoms. While the problem of delayed symptoms has been addressed for diagnosing hardware systems [33], this work generalizes the capability to more complex, software-extended behavior. This is accomplished by formulating the COP within a finite time horizon of N stages, where N is a parameter of the diagnostic system that controls the amount of history maintained. Given PHCA models and parameter N, the COP is generated automatically in an offline compilation phase. A novel approach, called N-stage Best-First Trajectory Enumeration (N-BFTE), is introduced to update and solve the COP in an online phase. N-BFTE dynamically updates the COP by shifting the time horizon and incorporating new observations while maintaining an N-stage history. The enumerated solutions correspond to the most likely trajectories of system evolution within the finite horizon.

- The third contribution is the application of state-of-the-art decomposition techniques [31, 24] to enhance the efficiency of the diagnostic process. By leveraging structural properties of the constraints during the offline compilation phase, the COP is decomposed into an equivalent acyclic instance. This allows the application of dynamic programming during the online phase and thereby the use of efficient decomposition-based optimization solvers as the computational core of N-BFTE.

- Finally, the diagnostic system depicted in Figure 1-2, is implemented and demonstrated on a number of scenarios, including vision-based rover navigation, models of the global metrology system of the MIT SPHERES [32] satellites, and payload models of the NASA New Millennium Earth Observing One spacecraft [28].

This thesis expands on preliminary versions published in [43, 42, 44].

1.5 Outline

The rest of this document is organized as follows.

Chapter 2 describes the PHCA modeling framework. The PHCA-based diagnosis problem is formally defined as N-Stage Best-First Trajectory Enumeration (N-BFTE), and illustrated for the vision-based navigation scenario.

The next two chapters describe the two phases of the novel diagnostic system: Chapter 3 introduces the formulation of N-BFTE for PHCA as a probabilistic constraint optimization problem (COP) during an offline compilation phase. The COP encodes the structure and execution semantics of PHCA as probabilistic constraints over a finite time horizon. The application of tree decomposition to subdivide the COP is described. Chapter 4 introduces the online monitoring and diagnosis algorithm that uses the offline COP formulation and the decomposed COP, to solve for the most likely PHCA state trajectories.

Chapter 5 presents the results of testing the implemented system on several representative scenarios, including vision-based rover navigation, and empirical validation on models of the MIT SPHERES satellite and the NASA Earth-Observing One spacecraft.

Chapter 2

Modeling and Monitoring Software-Extended Systems

This chapter reviews an expressive modeling framework, called Probabilistic Hierarchical Constraint-based Automata (PHCA) [52], which can support the specification of complex hardware and software behaviors. Building upon this framework, the problem of monitoring and diagnosing software-extended systems is formally defined as best-first enumeration of PHCA state trajectories over a finite time horizon. The process, referred to as N-Stage Best-First Trajectory Enumeration (N-BFTE), is demonstrated for the vision-based navigation scenario.

2.1 Modeling Software-Extended Behavior

Figure 2-1 shows the software-extended camera module for the vision-based navigation scenario described in Section 1.2. In this example, the failure probabilities for the battery, camera and sensor are 10%, 5% and 1%, respectively. A typical behavioral model of the camera is shown in Figure 2-2. The camera can be in one of 3 modes: on, off or broken. The hardware behavior in each of the modes is specified in terms of inputs to the camera, such as power, and the behavior of camera components, such as the shutter. The broken mode is unconstrained in order to accommodate novel types of failures. Mode transitions can occur probabilistically, or as a result of issued

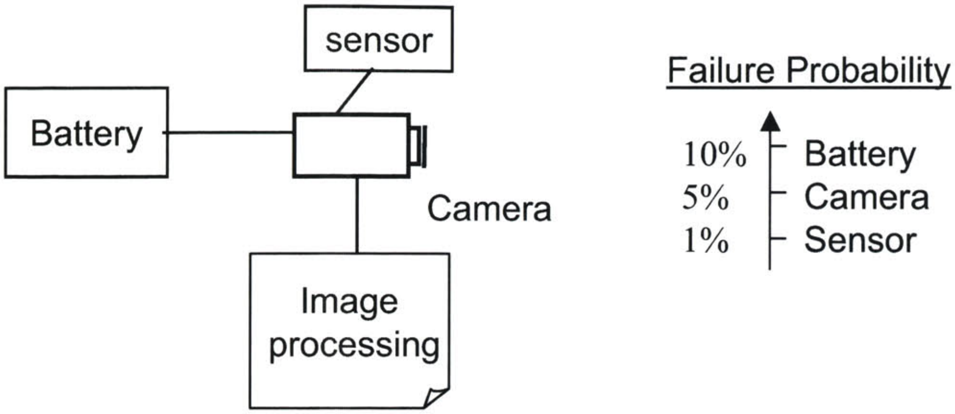

Figure 2-1: Camera module for navigation system

commands. The battery and the sensor components can be modeled in a similar way.

Given a scenario in which the sensor measures zero voltage at the camera, the most likely diagnoses of the module, generated based on the hardware models alone, are shown in Figure 2-3. However, the unmodeled software components can offer important evidence that substantially alters the diagnosis. A sample specification of

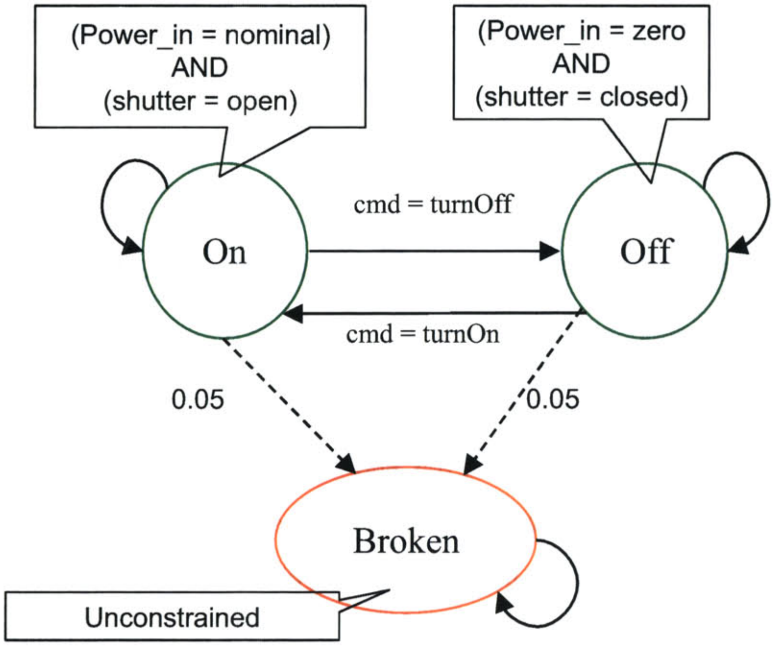

Figure 2-2: Behavioral model (PCCA) for the camera component.

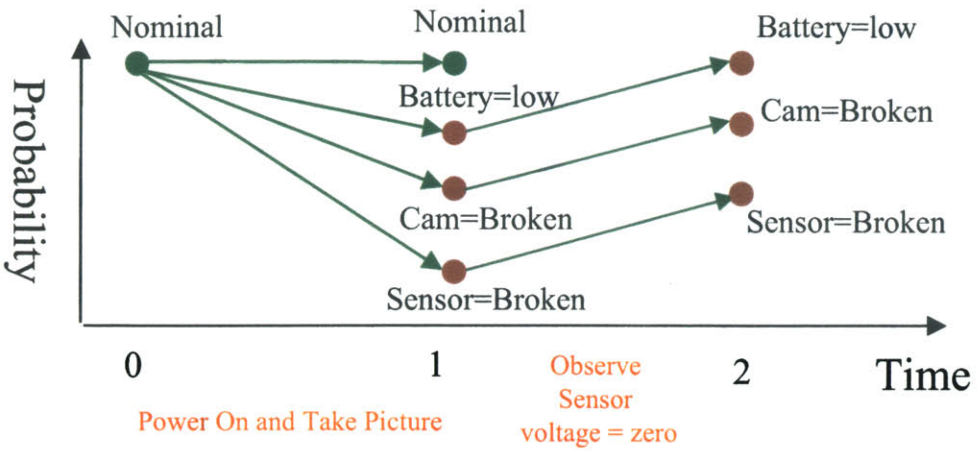

Figure 2-3: Most likely diagnoses of the camera module based on hardware component models. Nominal state = no failures.

the behavior of the image processing software may take the following form:

> *If an image is taken by the camera, process it to determine whether it is corrupt. If the image is corrupt, discard it and reset the camera; retry until a non-corrupt image is obtained for navigation. Once a high quality image is stored, wait for new image requests from the navigation unit.*

Such a specification abstracts the behavior of the image processing software implemented in an embedded programming language, such as the state-based RMPL language [52, 53]. For the above scenario, the behavior of the embedded software provides diagnostic information necessary to correctly estimate the state of the camera module. For example, given that the image is not corrupt, the possibility that the camera is broken becomes very unlikely. This is illustrated in Figure 2-4.

Typically, the behavior of a simple hardware component can be described by a small number of modes [16]. For diagnosing such components, a flat state machine representation, such as Probabilistic Concurrent Constraint Automata (PCCA) [54, 53] (see Appendix B), is adequate. However, software processes are generally more complex than simple hardware components. Therefore, a more structured, compact encoding of complex behavior is needed. Furthermore, diagnostic engines based on flat models such as PCCA, estimate each component to be in a single mode of be-

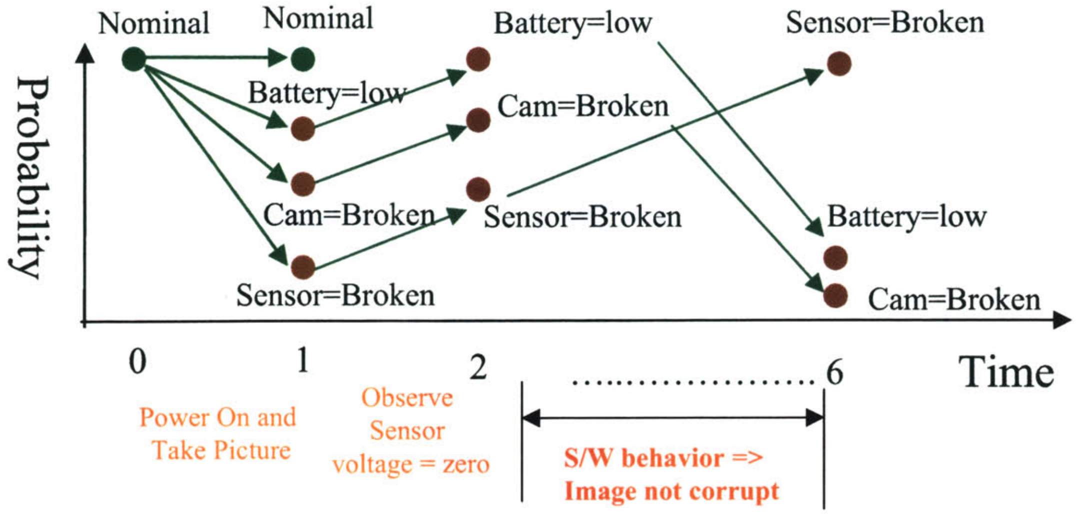

Figure 2-4: Most likely diagnoses of the camera module based on the software-extended behavior models.

havior. However, monitoring more complex software behavior necessitates tracking simultaneous hierarchical modes.

As highlighted in [52], a modeling formalism that will allow the specification of complex software and hardware behavior must support: 1) full concurrency for modeling sequential and parallel threads of behavior, 2) conditional behavior, 3) iteration, 4) preemption, 5) probabilistic behavior for modeling uncertainty and failure rates, and 6) constraints for specifying co-temporal relationships among variables. Requirements 1)-4) are key features of software processes; requirement 5) is essential for modeling stochastic processes for probabilistic reasoning; finally, requirement 6) is prevalent in qualitative modeling.

Probabilistic, Hierarchical Constraint Automata (PHCA) [52] have proven to be particularly effective representations for modeling complex behavior. Similar to Esterel [4], PHCA draw upon features of State Charts [27] - the foundation for the standard semantic model for most reactive embedded languages.

The following section reviews the PHCA modeling framework for handling requirements 1)-6), listed above.

2.2 Probabilistic, Hierarchical Constraint Automata

Probabilistic, hierarchical, constraint-based automata (PHCA) [52] are compact encodings of Hidden Markov Models (HMMs), and provide a mechanism for modeling both hardware and software behavior.

Definition 2-1 (PHCA)

A PHCA is a tuple $< \Sigma, P_\Theta, \Delta, O, C, P_T >$, where:

- Σ is a set of locations, partitioned into primitive locations Σ_p and composite locations Σ_c. Each composite location denotes a hierarchical, constraint automaton. A location may be marked or unmarked. A marked location represents an active mode of behavior.

- $P_\Theta(\Theta_i)$ denotes the probability that $\Theta_i \subseteq \Sigma$ is the set of start locations (initial state). Each composite location $l_i \subseteq \Sigma_c$ may have a set of start locations that are marked when l_i is marked.

- Δ is a set of finite domain variables. $C[\Delta]$ is the set of all finite domain constraints over Δ.

- $O \subseteq \Pi$ is the set of observable variables.

- $C : \Sigma \rightarrow C[\Delta]$ associates with each location $l_i \subseteq \Sigma$ a finite domain state constraint $C(l_i)$.

- $P_T(l_i)$, for each $l_i \subseteq \Sigma_p$, is a probability distribution over a set of transition functions $T(l_i) : \Sigma_p^{(t)} \times C[\Delta]^{(t)} \rightarrow 2^{\Sigma^{(t+1)}}$. Each transition function maps a marked location into a set of locations to be marked at the next time step, provided that the transition's guard constraint is consistent.

Definition 2-2 (PHCA State)

The state of a PHCA at time t is a set of marked locations, called a marking $m^{(t)} \subset \Sigma$.

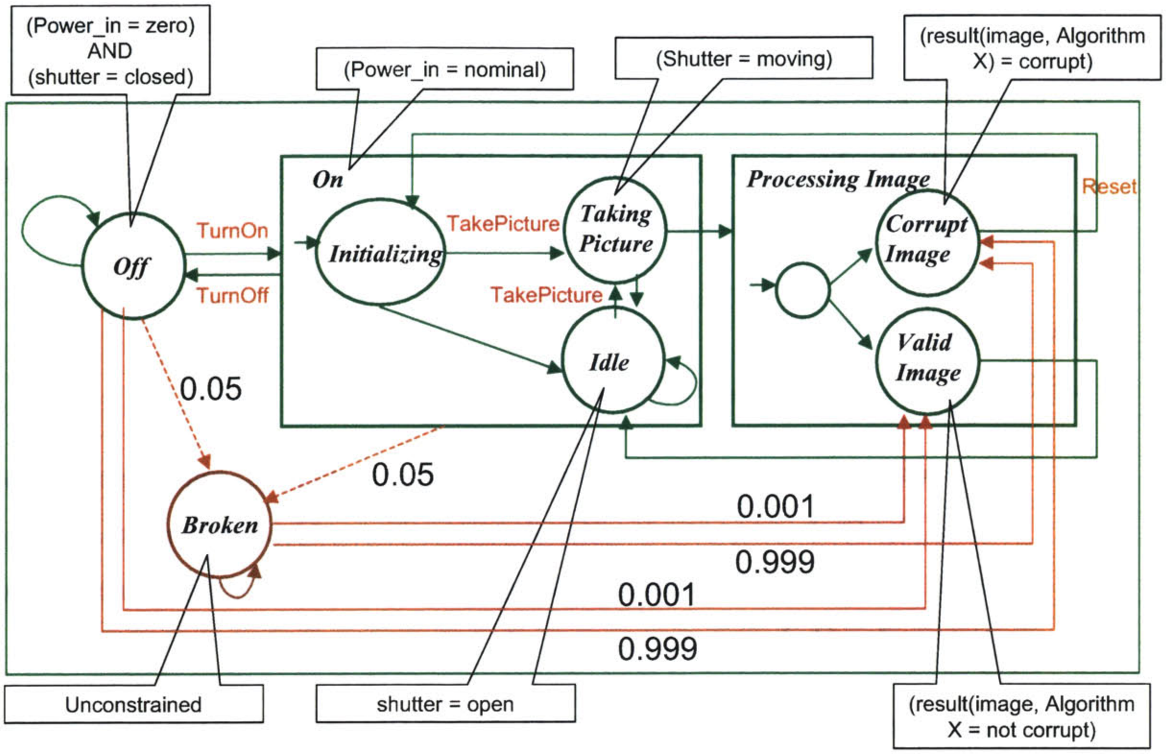

Figure 2-5: PHCA model for the camera/image processing module. Circles represent primitive locations, boxes represent composite locations and small arrows represent start locations.

Figure 2-5 shows a PHCA model of the camera module in Figure 2-1. The "On" composite location contains three subautomata that correspond to the primitive locations "Initializing", "Idle" and "Taking Picture". Each composite or primitive location of the PHCA may have behavioral state constraints, which hold as long as the automaton is in that location. The behavioral state constraint of a composite location, such as $(power\_in = nominal)$ for the "On" location, is inherited by each of the subautomata within that composite hierarchy. In addition to the physical camera behavior, the model incorporates qualitative software behavior, such as processing the quality of an image. Furthermore, based on the image quality, the possible camera configurations may be constrained by the embedded software. For example, if the image is determined to be corrupt, the software attempts to reset the camera. This restricts the camera behavior to transition to the Initializing location.

Recall that Figure 2-4 shows the most likely diagnoses based on the software-extended PHCA model. At time Step 2, as the sensor measurement indicates zero voltage, the most likely estimated trajectories end at 1) battery = low with 10% probability, 2) camera = broken with 5% probability and 3) sensor is broken with 1% probability. For the first trajectory, which indicates that the battery is low, the power to the camera is not nominal, hence the camera will stay in the "Off" location. For the second trajectory, the camera will be in the "Broken" location. For the third trajectory, which indicates that the sensor is broken, the power input to the camera will be unconstrained, and hence the PHCA state of the camera may include a marking of the "On" location. Although the evolutions of this third trajectory have an initially low probability of 1%, at time Step 6 they become more likely than the others, as the embedded software determines that the image is valid. The reason for this is that the second most likely trajectory at Time 2, with the camera = "Broken" location marked, has a 0.001 probability of generating a valid image. This makes the probability of that trajectory 0.005% at Time 6. This latter trajectory is less probable than those trajectories stemming from the sensor being broken with 1% probability. Similarly, the first trajectory with battery = low and camera = off becomes less likely at time Step 6, as there is 0.001% probability of processing a valid image while the camera is "Off".

PHCA models have the following advantages that support their use for diagnosing systems with software-extended behavior. First, since HMMs are intractable for encoding reactive systems, a compact PHCA encoding of HMMs is essential in order to support real-time, model-based deduction. This compact encoding is achieved by factoring HMMs into a set of concurrently operating automata. Second, PHCAs provide the expressivity to model the behavior of embedded software by satisfying requirements 1) to 6) described in the previous section. More specifically, PHCA satisfy these requirements through key features such as: concurrently active automata; state constraints associated with PHCA locations that must hold whenever a PHCA location is marked; hierarchical structure that enables modeling structured behaviors and the initiation and termination of complex concurrent and sequential behaviors; simultaneous marking of several transition targets, which enables modeling iterative and recursive behaviors. Further details of PHCA features can be found in [52].

As an example of concurrency, the PHCA in Figure 2-5 allows the simultaneous marking of the "On" location of the camera, as well as the "Initializing", "Idle", or "Taking Picture" locations. While an image-based navigation function may require high level camera state estimates, such as "On" or "Off", a function that coordinates imaging activities may need more detailed camera state estimates, such as "Initializing" or "Taking Picture". Simultaneous marking of several camera locations such as "On" and "Initializing", allows their use within functions that require estimates at different levels of granularity. This is in contrast to diagnosis based on non-hierarchical models that estimate each component to be in a single mode of operation.

The next section formally defines the problem of PHCA-based monitoring and diagnosis in the presence of delayed symptoms, thereby establishing the framework for addressing the challenges set forth in Section 1.3.

2.3 Best-First Trajectory Enumeration for PHCA

PHCA-based monitoring and diagnosis are each formulated as the task of enumerating and tracking the most likely trajectories of system state. Given a PHCA state distribution at time t and an assignment to observable and command variables in Δ (see Definition 2-1) at times $t+1$ and t respectively, **Best-First Trajectory Enumeration** (BFTE) is the problem of estimating the most likely transitions to PHCA states at time $t+1$. BFTE is a best-first shortest path problem, which can be solved using a variant of the Viterbi algorithm [30], to find the most likely hypotheses that explain the observations and commands.

Maintaining all possible state trajectories at each time step is intractable, because of the exponential growth in state space. Diagnostic systems have achieved reactivity by typically focusing on the leading diagnoses [13, 15, 16, 54, 53]. For instance, at every time step, the Titan mode estimation engine [53] maintains a limited number of trajectories (K-Best). A potential problem with this approach is that it may miss the best diagnosis. This would occur if a trajectory through a state is initially very

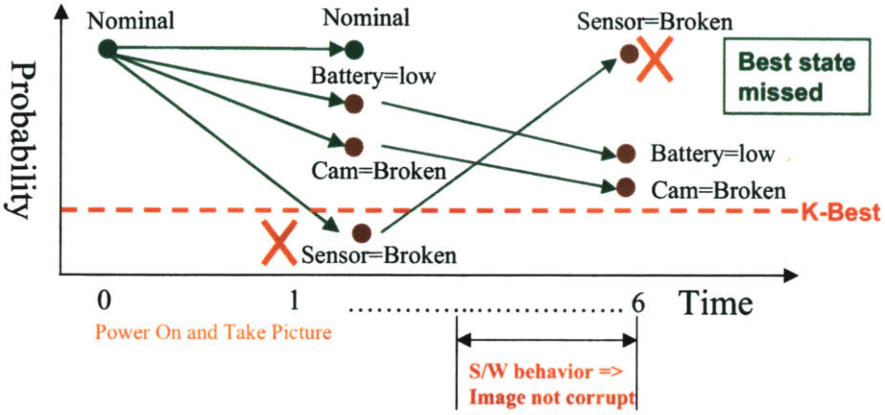

Figure 2-6: Missed diagnosis as a result of tracking a limited number of trajectories (K-Best)

unlikely and hence is pruned; however, it becomes very likely after additional evidence. Figure 2-6 illustrates this situation for the camera module, where the initially unlikely state ($Sensor = Broken$) is pruned. This results in the best diagnosis to be unreachable when additional evidence is available at Time 6.

Dealing with delayed evidence is particularly important for diagnosing systems with software-extended behavior, due to typically delayed observations associated with software processing. The Livingstone-2 (L2) engine [33] addressed the problem of delayed symptoms for diagnosing systems modeled as concurrent constraint automata. This work generalizes delayed-symptom diagnosis to PHCA, by performing BFTE within a receding, N-Stage time horizon. Unlike L2, the N-BFTE approach limits the number of trajectories being tracked, while leveraging the N-Stage history of observations and issued commands to reason about delayed symptoms. This approach avoids a large increase in the number of trajectories tracked by L2 when inconsistencies arise due to delayed symptoms. The L2 approach and comparison to this work were discussed under related work in Section 1.3.

2.4 N-Stage Best-First Trajectory Enumeration

Consider the problem of **N-Stage Best-First Trajectory Enumeration** (N-BFTE) for PHCA. Given a probability distribution $P(S^{(t)})$ of PHCA states S at time t, and assignments to observable and command variables in Δ at times $[t..t+N]$ and $[t..t+N-1]$, respectively, the task of N-BFTE is to estimate the most probable trajectories, within the time horizon $[t..t+N]$, that are consistent with the observations, commands and the PHCA model. The probability of a trajectory $< S^t, S^{t+1}, S^{t+2}, .., S^{t+N} >$ is defined as:

$$P(S^t) \cdot \prod_{j=0..N-1} P_T(S^{t+j+1}|S^{t+j}, \Delta^{t+j}) \cdot \prod_{j=0..N} P(O^{t+j}|S^{t+j}) \tag{2.1}$$

where P_T is the transition probability from a current state to a target state. A PHCA state consists of multiple marked locations, each of which may transition to multiple target locations. Hence, T is encoded as a set of transitions taken from locations within the current state and leading to locations within the target state. Therefore, P_T is the product of the probabilities of all transitions $\tau \in T$. In this thesis the probability of observations O^{t+j} for $j = 0..N$ in Equation 2.1 is taken to be:

$$P(O^{t+j}|S^{t+j}) = \begin{cases} 1 & if\ O^{t+j} \wedge S^{t+j}\ consistent \\ 0 & otherwise \end{cases} \tag{2.2}$$

Note that Equation 2.2 is an approximation to the observation probabilities. Consider the case in which an assignment to an observable variable o_i is not entailed, but there exist values of o_i that are consistent with the current state S. Then, as discussed in [15], a uniform distribution over the possible values of o_i can typically be assumed. This means that the probability $P(o_i|S) = 1/m$, where m is the number of consistent values of o_i. Equation 2.2 is thus an overestimate of this uniform distribution. However, the uniform distribution introduces the complexity of counting additional satisfying assignments. In [37], a method for incorporating an observation probability distribution is devised.

The same approximation applies to the guard constraints of PHCA transitions (see Definition 2-1). Recall that P_T in Equation 2.1 refers to the probability of the set of enabled transitions T. A necessary condition for enabling each transition $\tau \in T$ is that its guard is satisfied. Similar to observation probabilities, Equation 2.1 implicitly assumes that the transition guard probabilities are either 0 or 1. The probability is 0 if the guard is inconsistent with observations and commands, and 1 otherwise. This reduces P_T to the product of probabilities of consistent transitions. However, consider the case in which a guard constraint is consistent with observations and commands, but not entailed. Then, the guard probability 1 is an overestimate. A better probability estimate can be obtained by counting the number m of models that are consistent with a given state S, and the number s of subsets of those models that satisfy the guard. Then, the probability of the guard is s/m. However, as mentioned above, this process is computationally expensive, and thus an approximation is used in practice.

The likelihood of the trajectory defined in Equation 2.1 is based on the following two key assumptions. The first assumption is that transitions are conditionally independent, given the prior state S. This is analogous to component failure independence assumptions made in previous diagnostic systems [15, 16, 54]. This assumption enables the transition probability P_T to be expressed as the product of probabilities of individual transitions $\tau \in T$, for each time step within the finite time horizon. Second, the observation and transition guard probabilities are assumed to be 1.0 if they are consistent with the current state. The limitation of this is that it does not provide a bias towards observations and guards that are entailed, rather than simply consistent. On the other hand, this approximation avoids the added complexity of counting all consistent interpretations.

Generating the maximum probability trajectories within the time horizon differs from approaches that calculate a belief state [52, 38]. A belief state is a probability distribution over possible states of the system. Figure 2-7(a) shows a *Trellis Diagram* [53], which represents the evolutions of system state. Notice that each state can be reached through multiple previous states. Therefore, belief state calculation requires

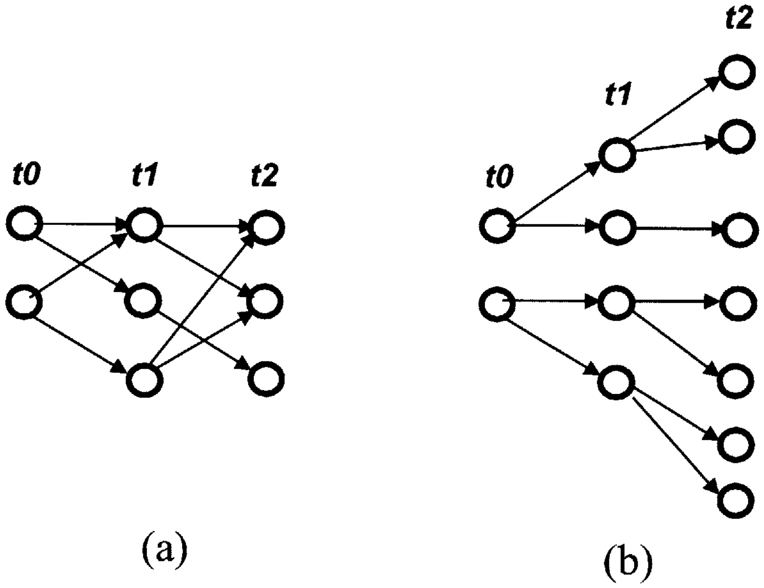

Figure 2-7: (a) Trellis diagram showing the evolution of an approximate belief state; (b) branching tree representation showing state trajectories.

calculating the probability of each state by summing all transition probabilities that lead to that state. This is accomplished by using the belief state update equations for hidden Markov models [3]. Since full belief state maintenance is intractable for many real world applications, in practice only an approximation is computed, based on a limited number of maintained states.

The N-BFTE approach generates the most likely trajectories, rather than a belief state, by branching trajectories outward at every time step, as shown in Figure 2-7(b). This approach is referred to as *Maximal Probability Diagnosis* [47, 12].

Chapters 3 and 4 further elaborate the discussion on the likelihood of the trajectories defined in this section. The following section gives an overview of the two-phase approach to N-BFTE.

2.5 The N-BFTE Process

The N-BFTE process, as highlighted in Chapter 1 and illustrated in Figure 1-2, consists of two phases: an offline compilation phase and an online solution phase.

The steps of the process are as follows.

2.5.1 Offline Phase

The offline phase occurs before runtime, and does not require the availability of any observations or commands. This phase consists of three steps:

- **PHCA Specification:** Mixed software and hardware behavior is specified in the state-based specification language RMPL, and is compiled into PHCA. The PHCA models are represented graphically, as shown in Figure 2-5. The compilation of RMPL to PHCA is presented in [52].
- **PHCA Encoding:** Given a PHCA model, the N-BFTE problem for PHCA is formulated as a soft constraint optimization problem (COP) within the N-Stage horizon. The soft constraints encode the PHCA models and their execution semantics over the N-Stage horizon. This encoding does not require the availability of observations and commands. As discussed above, the N-BFTE formulation enables accounting for delayed symptoms during diagnosis. Furthermore, encoding the PHCA semantics combines the model simulation and consistency checking steps during the online diagnosis process.
- **Tree Decomposition of COP:** The COP generated in the PHCA encoding step is decomposed into a tree structure. This removes cycles from the constraint network, thus enabling efficient solution during the online phase.

The benefit of the offline phase is that the models are compiled into a constraint optimization problem, and decomposed into an equivalent acyclic instance prior to runtime. This enables the efficiency of the online monitoring and diagnosis phase.

2.5.2 Online Phase

The online phase corresponds to runtime, when observations and commands are available. The COP formulated and decomposed in the offline phase is dynamically updated to incorporate observations and commands. The updated COP is solved using a decomposition-based constraint solver [47, 48]. The solutions to the COP are enumerated in best-first order, thus solving the N-BFTE problem within each time horizon. The solutions correspond to the K-Best PHCA trajectories in decreasing order of probability, as given by Equation 2.1. As the time horizon shifts, the COP is updated to track the trajectories obtained from the previous horizon, and the N-BFTE process repeats.

The online phase uses decomposition-based COP solvers to efficiently compute the system diagnoses. The online phase achieves the best performance for a highly structured system that can be optimally decomposed in the offline phase.

2.6 Summary

This chapter reviewed probabilistic, hierarchical constraint automata (PHCA) as a modeling framework for complex systems. The relative advantages of PHCA over previous models were highlighted in the context of modeling and diagnosing software-extended systems. In particular, the vision-based navigation scenario, introduced in Section 1.2, was modeled as a PHCA. Based on this PHCA model, diagnosis in the presence of delayed symptoms was illustrated. Diagnosis was defined as the Best-First Trajectory Enumeration (BFTE) of PHCA state trajectories. In order to account for delayed symptoms, the BFTE problem was framed over an N-Stage time horizon (N-BFTE). Within the N-BFTE framework, the likelihood of a PHCA state trajectory was defined in terms of the initial state probability of the PHCA, the state transition probabilities along the trajectory, and observation probabilities in each state within the trajectory. The solutions to N-BFTE thus correspond to the most likely trajectories within the N-Stage horizon. N-BFTE is thus an instance of *maximal probability diagnosis*. Finally, a two-phase approach to tackling the N-BFTE problem was introduced, building upon the diagnostic system architecture introduced in Chapter 1. In the offline phase, the N-BFTE problem is framed as a soft COP, which encodes the semantics of the PHCA. This effectively unifies the model simulation and consistency checking steps in the online phase. Furthermore, the COP is decomposed into an acyclic instance, motivated by recent developments in decomposition-based optimal constraint solvers. In the online phase, the COP is updated to incorporate runtime observations and commands, and is solved efficiently using a decomposition-based solver. The following two chapters describe each of the offline and online phases, respectively.

Chapter 3

Offline Phase: Formulation of COP

The PHCA-based monitoring and diagnosis process consists of two phases: offline model compilation and online execution. This chapter focuses on the offline phase, which occurs before commands and observations are available. The models are first compiled into a constraint optimization problem (COP) over a finite time horizon, which is then decomposed into an acyclic instance.

3.1 Encoding the N-BFTE Problem for PHCA

The N-BFTE problem for PHCA models, described in the previous chapter, is formulated as a soft COP [50, 2], which associates a value with each constraint assignment. As discussed in Section 1.3, previous work [55, 47, 53] has framed diagnosis as a COP. However, the approach in this thesis is distinguished collectively by three features. First, the COP is formulated over a finite time horizon, thereby enabling diagnosis in the presence of delayed evidence. Second, the COP encodes the structure and execution semantics of PHCA models. This encoding is particularly convenient for reasoning over several time steps, since it collapses the model simulation and consistency with observations and commands into the single task of solving a constraint optimization problem (COP) over the finite horizon. The soft constraint framework offers convenient expressivity for encoding the models by associating probability values with arbitrary constraints, rather than just variables to be solved for [50, 2].

Finally, the soft constraint framework enables leveraging an extensive body of recent advances in decomposition-based optimal constraint solvers [47, 51, 48], in order to efficiently compute the solutions.

This chapter presents the full encoding of PHCA models as a soft COP for performing N-BFTE, and the subsequent decomposition of the COP into independent subproblems. The following starts by introducing the soft constraint framework, and in particular the special case of valued constraints.

3.1.1 Soft and Hard Constraints

Constraint programming [36, 17] is an approach for solving combinatorial problems. This is accomplished by framing the problem as a set of variables with corresponding finite domains representing sets of possible values for the variables, and constraints that restrict the possible combinations of values of the variables. Combinations of values to variables are generally referred to as tuples. A solution to such a constraint problem is an assignment to variables of interest - referred to as solution variables - that satisfy all constraints. These type of problems are thus referred to as constraint satisfaction problems (CSP) [17]. Furthermore, a class of problems, called Optimal CSPs (OCSP) [17], additionally specify a multi-attribute utility function over solution variables (referred to as decision variables in this context). The solution to an OCSP must thus satisfy all constraints, while simultaneously maximizing (or minimizing) the utility function, defined over the decision variables.

Within the CSP framework, the constraints take the form of *hard constraints*, that is, tuples are either allowed by the constraint or not. However, there are problems for which the hard constraint formulation is inadequate, such as problems that must deal with uncertainty. *Soft constraints* [50] provide a generic framework for problems that must incorporate uncertainty, fuzziness, partial or preferential constraint satisfaction. There are various types of soft constraint problems, as surveyed in [2]. This work is based on a particular class of soft constraint problems, called valued constraint satisfaction problems (VCSP), which associate valuations with constraints [50]. In particular, a probabilistic CSP is a special case of a VCSP in which the constraint valuations correspond to probabilities [22].

The following formally defines the COP as a VCSP [50], and in particular a probabilistic CSP [22].

Definition 3-1 (Constraint Optimization Problem)
A *constraint optimization problem* (COP) consists of a set of variables $X = \{X_1, ..., X_n\}$ with a corresponding set of finite domains $D = \{D_1, ..., D_n\}$, a set of constraints $R = \{R_1, ..., R_m\}$, a valuation structure $(E, \leq, \oplus, \bot, \top)$, and functions $F = \{F_1, ..., F_m\}$ mapping tuples of the constraints R to E. Each constraint $R_i \subseteq D_{i1} \times ... \times D_{ik}$ is defined over variables $\{X_{i1}, ..., X_{ik}\} \subseteq X$. The set E is totally ordered by $\leq$ with a minimum element $\bot \in E$ and a maximum element $\top \in E$, and $\oplus$ is an associative, commutative, and monotonic operation, with identity element $\bot$ and absorbing element $\top$. Each $F_i \in F$ is a function $F_i : R_i \rightarrow E$, mapping tuples of R_i to values in E. Functions F are thus called valuations of constraints R.

The set of values E allows different levels of constraint satisfaction to be expressed. The element $\bot$ means that the constraint is satisfied, and $\top$ means that the constraint can never be violated. Within this valuation framework, a constraint is hard if all its valuations are either $\bot$ or $\top$.

A probabilistic CSP [22] is a special case of VCSP with a valuation structure $(E, \leq, \oplus, \bot, \top) = ([0, 1], \max, \cdot, 1, 0)$, as described in [5]. Values in E correspond to probabilities ordered by max, or equivalently $\geq$. Probability 1.0 is associated with constraint tuples that hold with certainty and probability 0.0 is associated with constraint tuples that are not allowed by the constraint. Finally, $\cdot$ corresponds to multiplication over the real numbers.

Given the COP in Definition 3-1, the optimal solution to the COP is an assignment $t \in D_1 \times ... \times D_n$ to variables X that has the best value obtained by aggregating the valuations F. The operation $\oplus$ is used to combine several valuations, such that the value V of t, an assignment to X, is given by:

$$V(t) = (\bigoplus_{i=1}^{m} F_i)(t). \tag{3.1}$$

Suppose that the solution to the COP is desired for a subset $Y \subseteq X$ of variables of interest. Then, in addition to the *combination* operator $\oplus$, *projection* of the valuations onto the subset Y of variables is required. Combination and projection are formally defined as follows [17, 49]:

Definition 3-2 (Combination and Projection)

Let f, $g \in F$ be two valuation functions. Let $t \in D_1 \times ... \times D_n$, and let $t \downarrow_Y$ denote the restriction of an assignment t to a subset $Y \subseteq X$ of variables. Then:

1. The *combination* of f and g, denoted $f \oplus g$, is the valuation function that maps each t to the value $f(t) \oplus g(t)$;

2. The *projection* of f onto a set of variables Y, denoted $f \Downarrow_Y$, is the valuation function that maps each t to the value $f(t_1) \leq f(t_2) \leq \ldots \leq f(t_k)$, where $t_1, t_2, \ldots, t_k$ are all the assignments for which $t_i \downarrow_Y = t$. Recall that $\leq$ is the total order operator in the valuation structure $(E, \leq, \oplus, \perp, \top)$.

Therefore, given the COP in Definition 3-1 and a subset $Y \subseteq X$ of variables of interest, the optimal solution to the COP is an assignment t to variables Y that has the best value given by:

$$V(t) = ((\bigoplus_{i=1}^{m} F_i) \Downarrow_Y)(t) \tag{3.2}$$

Based upon the soft constraint framework presented above, the following sections introduce the COP formulation of the N-Stage best-first trajectory enumeration (N-BFTE) problem for PHCA. Furthermore, the functions F_i are defined such that the solutions to the COP, which have values given by Equation 3.2, correspond to the most likely PHCA state trajectories defined by Equation 2.1.

3.1.2 Elements of the COP Formulation

In order to perform N-Stage best-first trajectory enumeration for PHCA, the structure and semantics of a PHCA model is encoded as a soft COP (Definition 3-1) that consists of:

- A set of variables $L_\Sigma^t \cup \Delta^t \cup E^t$ for each of $t = 0..N$, where $L_\Sigma^t = \{L_1^t, ..., L_n^t\}$ is a set of variables that correspond to PHCA locations $l_i \in \Sigma$, Δ^t is the set of PHCA variables at time t, and $E^t = \{E_1^t, ..., E_n^t\}$ is a set of auxiliary variables, used to encode the execution semantics of the PHCA within the N-Stage time horizon.

- A set of finite, discrete-valued domains $D_{L_\Sigma} \cup D_\Delta \cup D_E$, where $D_{L_\Sigma} = \{Marked,$ $Unmarked\}$ is the domain for each variable in L_Σ, D_Δ is the set of domains for PHCA variables Δ, and D_E is a set of domains for variables E.

- A set of constraints R that encode the PHCA and their execution semantics over the N-Stage time horizon. For instance, these constraints specify initial PHCA marking, state behavior and transition consistency, and the semantics of stepping the PHCA, by identifying enabled transitions and the resulting PHCA markings over several time steps. Types of constraints R and their complete formulation are presented in the following sections.

- A probabilistic valuation structure $(E, \leq, \oplus, \perp, \top) = ([0,1], \max, \cdot, 1, 0)$, as discussed in the previous section [5]. The choice of the total order max favors higher probability values, while the choice of multiplication $\cdot$ as the combination operator relies on conditional independence of the component probabilities. This form of diagnosis is referred to as *Maximal Probability Diagnosis* [47, 23, 12].

- A set of functions $F : R \rightarrow [0,1]$ that map tuples of constraints R to probabilities. For hard constraints, tuples disallowed by R are assigned probability 0; tuples allowed by R are assigned probability 1. For soft constraints, tuples are mapped to a range of probability values $\in$ [0,1], based on the PHCA model, as given by functions F presented below. These functions incorporate the probability distribution P_Θ of PHCA start states, and probabilities P_T associated with PHCA transitions (refer to Definition 3-1).

- The solution variables of the COP are L_Σ^t for $[t..t + N]$; each assignment to the solution variables represents a PHCA state trajectory, where each state is a marking of PHCA locations. The optimal solution to the COP is an assignment T to the solution variables that has the maximum valuation V_{Best}, given by:

$$V_{Best} = \max_T[((\prod_{i=1}^{m} F_i) \Downarrow_{\{L_\Sigma^t ... L_\Sigma^{(t+N)}\}})(T)] \tag{3.3}$$

$$T = \arg\max_T[((\prod_{i=1}^{m} F_i) \Downarrow_{\{L_\Sigma^t ... L_\Sigma^{(t+N)}\}})(T)] \tag{3.4}$$

The aggregate valuation V_{Best} in Equation 3.3 is obtained by instantiating Equation 3.2 for the probabilistic valuation structure $([0, 1], max, \cdot, 1, 0)$. The projection operator $\Downarrow$ is maximization, as presented in Definition 3-2. This means that the best solution is picked such that its extension to all variables generates the maximum probability value. This formulation results in an instance of *maximal probability diagnosis* [47, 23, 12]. Note that the multiplication operator $\cdot$ for combination appears as a product $\prod$ in Equation 3.3. The functions F_i in Equation 3.3 are introduced in the next sections, such that the value of the best solution V_{Best} corresponds to an upper bound on the maximum probability of the trajectory defined by Equation 2.1. As pointed out in [12], the probabilities are not necessarily exact; they are only approximate values, used to identify the more plausible diagnoses.

Recall that enumerating the most likely trajectories (BFTE) differs from that of belief state enumeration [38, 52]. State enumeration considers all transitions that may lead to the same state. The belief state is then computed using the standard hidden Markov model (HMM) belief state update equations [3]. This involves summing transition probabilities from previous states to the current state. In practice, tractability in this framework is achieved by computing an approximation to the true belief state, typically by limiting the number of states maintained. In contrast to belief state enumeration, transition probabilties are not summed in the case of N-BFTE. This corresponds to an instance of the Viterbi algorithm [30].

3.1.3 Encoding the PHCA Execution Semantics

A key to framing PHCA-based N-BFTE as a COP is the formulation of the constraints R that capture the behavior and execution semantics of the PHCA, and the functions F that map the constraints R to probability values that capture the uncertainty within the PHCA models.

PHCA execution involves determining the consistency of state behavior constraints, identifying enabled transitions from a current PHCA state, and taking those transitions to determine the next state. In the following, nested parentheses are used to represent levels of hierarchy in a state assignment. For example (On(Idle)) means the state is in the "Idle" location within "On". Referring back to the PHCA example in Figure 2-5, if at time t the PHCA state is $(On(Idle))$ and the transition guard constraint $(command = TakePicture)$ is satisfied, and at time t+1 the state constraint $(shutter = moving)$ of the transition's target location is consistent, then the PHCA state at time t+1 will be $(On(TakingPicture))$.

The following semantic rules apply to PHCA hierarchies:

- **Full Marking of Descendent Start Locations:** When a composite location becomes marked, all of its start locations (subautomata) are marked. For example, since "Initializing" is a descendent start location of the composite "On" location (as indicated by small arrows in Figure 2-5), a PHCA in state "Off" may transition to state $(On(Initializing))$.

- **Hierarchical Composite Marking/Unmarking:** A composite location should be marked if any of its subautomata are marked, and unmarked if none of its subautomata are marked. For example, in the camera model, marking the "Idle" location necessitates marking the "On" location. Furthermore, the "On" location can not be marked if the camera is not initializing, idle or taking picture.

The semantic rules for PHCA execution are captured via constraints presented below. In particular, there are two types of full marking constraints that impose the conditions for a full marking. One for encoding the initial (at t=0) PHCA state as probabilistic full markings of locations, and the other for encoding full markings of transition targets. The first imposes the full marking for $t = 0$, based on the initial state distribution P_{Θ}. The second defines marking for all $t > 0$, when transitioning from one state to the next. Each of these is introduced in the next section, under T=0 constraints and transition constraints, respectively.

The complete set of constraints R within the COP formulation are divided into four categories:

- **Consistency constraints** specify the consistency of PHCA state constraints and transition guard conditions with commands and observations.

- **T=0 constraints** limit the initial PHCA states to a probabilistic distribution on start states. These constraints are applicable to the initial time $t = 0$ only.

- **Transition constraints** encode the semantics of consistent transitions and probabilistic choice among consistent transitions, as well as identification and full marking of transition targets.

- **Marking constraints** encode the conditions for marking primitive and composite PHCA locations, including the hierarchical composite marking rule, stated above.

3.1.4 Consistency Constraints

The PHCA execution semantics builds upon the consistency of state constraints and transition guards, with respect to the current state marking, commands and observations. Variables E_T, with domain $\{Consistent, Inconsistent\}$, are introduced to indicate the consistency of each state constraint and transition guard at each time point. Note that these consistency variables are introduced in order to facilitate the encoding of more complex constraints, described in the following sections. In particular, using these variables within a more complex constraint reduces the number of variables that appear in the scope of that constraint.

Consistency constraints are generated for all locations with associated state constraints and for all transitions that have guards.

State Constraint Consistency:

These constraints encode the consistency of PHCA state constraints with respect to the current state marking and observations, for each time t.

$$\forall t \in \{0..N\}, \forall L \in \Sigma : State^t_L = Consistent \Leftrightarrow C(L)^t \tag{3.5}$$

where $State^t_L \in E$ is a variable introduced to denote the consistency of state constraints with observations. The constraint in Equation 3.5 is instantiated for each primitive and composite PHCA location L that has a behavioral state constraint C(L) (see Definition 2-1). Recall that $C(L)$ is a propositional logic sentence that specifies the behavior of the PHCA when location L is marked. Referring to the camera model in Figure 2-5, the "Off" location has a conjunctive constraint $C("Off")$ $\equiv$ $(power_{in} = zero \wedge shutter = closed)$.

The state constraint consistency, in Formula 3.5, is a hard constraint, that is, it maps to tuples with probability value 1.0 if the tuples are consistent with the constraint; otherwise, the tuple has value 0.0. The tuples of the constraint are generated by enumerating sets of assignments to observable variables (interpretations) that are models of the state constraint, that is, they satisfy the state constraint. Table 3.1 shows the tuples of the constraint with corresponding probability values for the "Off" location of the camera model. Note that the domains of the $power_{in}$ and $shutter$ variables are $\{zero, nominal\}$ and $\{open, closed, moving\}$, respectively.

$State^t_{Off}$	$power^t_{in}$	$shutter^t$	**Probability**
Consistent	{zero}	{closed}	1.0
Inconsistent	{nominal}	{open, moving}	1.0

Table 3.1: State consistency for camera "Off" location, for generic time t.

$Guard^t_{\tau}$	$command^t$	**Probability**
Consistent	{TakePicture}	1.0
Inconsistent	{TurnOn, TurnOff, Reset, No-command}	1.0

Table 3.2: Guard consistency constraint for $\tau \equiv (Initializing \rightarrow TakingPicture)$, for generic time t.

Transition Guard Consistency:

Similar to the state constraints, the transition guard consistency constraints, given by Formula 3.6, encode the consistency of transition guards with respect to issued commands and observations.

$$\forall t \in \{0..N-1\}, \forall \tau \in T : Guard^t_{\tau} = Consistent \Leftrightarrow C(\tau)^t \qquad (3.6)$$

$C(\tau)$ is a propositional logic sentence, referred to as $C[\Delta]$ in Definition 2-1. Within the camera model, the commands *TurnOn, TurnOff, TakePicture* and *Reset* are all transition guards. The consistency of a given transition guard with observations and issued commands is a necessary, but not sufficient condition for that transition to be taken. Consider the transition guard *TakePicture*, used to transition from the *Initializing* location to the *TakingPicture* location. This is given by $C(Initializing \rightarrow TakingPicture) \equiv (command = TakePicture)$. For each time t, this guard is mapped to a constraint, as shown in Figure 3.2. In this example, the *command* variable has the domain {*TurnOn, TurnOff, TakePicture, Reset, No-command*}.

Mapping Propositional Formulae to Valued Constraints

For each consistency constraint, the constraint's tuples are generated by first converting the propositional sentences $C(L)$ and $C(\tau)$ to disjunctive normal form (DNF). This form corresponds to a disjunction of conjunctive propositional clauses. Models of the conjunctive clauses in the DNF sentence are equivalent to tuples of the constraint that are associated with the assignments *Consistent*. Tuples associated with *Inconsistent* assignments are generated by first negating the original propositional sentence, and then converting this negation into DNF from which the tuples are derived as described.

For example, consider the state constraint $C(L) \equiv (X = a \vee Y = b)$, where the domain of each of X and Y is $\{a, b\}$. This disjunctive clause is equivalent to a DNF sentence, consisting of the two "conjunctive" clauses $X = a$ and $Y = b$. Each of these clauses is equivalent to a tuple associated with the assignment *Consistent*. That is, $X = a$ is equivalent to the first tuple, in which the values of Y are not constrained. $Y = b$ is equivalent to the second tuple, in which the values of X are not constrained. In order to get the tuples associated with *Inconsistent*, the negation of $X = a \vee Y = b$ is first computed as $not(X = a) \wedge not(Y = b)$. This is a DNF sentence with a single clause $not(X = a) \wedge not(Y = b)$. The clause maps directly to the tuple $X = b$ and $Y = a$, which is associated with the assignment *Inconsistent*.

3.1.5 T=0 Constraints

The following four constraints apply only to the initial time T=0, and will be referred to as T=0 Constraints. These constraints enable the probabilistic marking of PHCA start locations. More specifically, if the state constraint of a starting location is not consistent with initial observations, it will not be marked; otherwise, it will be marked probabilistically. On the other hand, non-starting locations are initially unmarked. For each marked composite location, all of its descendent start locations must be marked, recursively, based on the full marking semantics discussed previously. This initial marking of start states is probabilistic.

T=0 Model Marking:

This constraint specifies that each top-level PHCA is started at $t = 0$, and is formulated as:

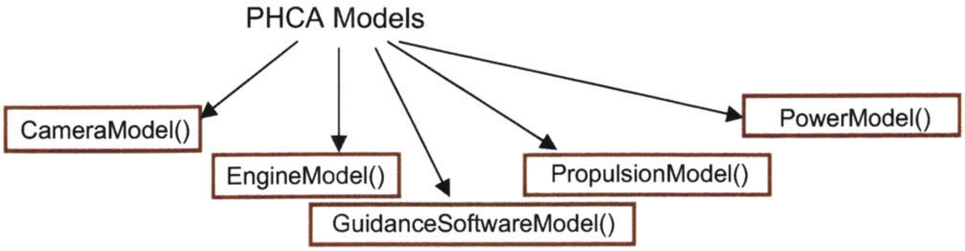

Figure 3-1: PHCA models for spacecraft subsystems.

$Camera^0$	$Engine^0$	$GuidanceSoftware^0$	$Propulsion^0$	$Power^0$	Probability
Marked	Marked	Marked	Marked	Marked	1.0

Table 3.3: Model Marking at T=0

$$\forall C \in (PHCAModels) : C^0 = Marked \tag{3.7}$$

In Constraint 3.7, PHCA Models refers to a set of individual PHCA that model top-level subsystems. For example, a spacecraft model may contain a PHCA model for each of its five subsystems (Figure 3-1. The camera model in Figure 2-5 is one instance of a PHCA model.

For the example in Figure 3-1, Table 3.3 specifies the constraint that all PHCA models are marked at $t = 0$. Note that the models will stay marked for $t > 0$ as long as at least one location (subautomaton) within the model is marked. This holds because, based on the hierarchical composite marking semantics, the PHCA model will be marked.

T=0 Unmarking:

This constraint specifies that all non-start PHCA locations are *Unmarked* at $t = 0$.

$$\forall C \in \Sigma_C, \forall L \in (Subautomata(C) - \Theta(C)) : L^0 = Unmarked \tag{3.8}$$

For the camera model in Figure 2-5, each of the locations *TakingPicture, Idle, ProcessingImage, CorruptImage* and *ValidImage* are initially unmarked.

$Camera^0$	$Start^0_{Off}$	$Start^0_{On}$	$Start^0_{Broken}$	**Probability**
Marked	Enabled	Enabled	Enabled	1.0
Unmarked	Disabled	Disabled	Disabled	1.0

Table 3.4: T=0 Full Marking constraint for the camera in Figure 2-5.

T=0 Full Marking:

This constraint enforces the initial full marking of composites, that is, the start locations of each marked composite automaton are marked. This is formulated as:

$$\begin{aligned}\forall C \in \Sigma_C : [C^0 = \mathit{Marked} \Leftrightarrow (\forall L \in \Theta(C) : \mathit{Start}^0_L = \mathit{Enabled})] \\ \wedge[C^0 = \mathit{Unmarked} \Leftrightarrow (\forall L \in \Theta(C) : \mathit{Start}^0_L = \mathit{Disabled})]\end{aligned} \tag{3.9}$$

In this formulation, $Start_L$ is a variable associated with start location L, taking on values from the domain {*Enabled, Disabled*}. This variable is introduced because start locations can not be explicitly marked, since their state constraint may be violated at $t = 0$. Instead, a *Start* variable associated with each start location is enabled. This variable is then used to probabilistically mark those start locations that have consistent state constraints, according to the initial state distribution for the PHCA. The probabilistic marking of start locations is implemented by the initial probabilistic marking constraint, introduced in the next section.

Constraint 3.9 is instantiated for each composite location that contains start locations. As an example, consider the PHCA for the camera model. This PHCA constitutes a composite location containing start subautomata *Off, On* and *Broken.* The *On* location itself is a composite location that contains the subautomaton *Initializing.* Likewise, the *ProcessingImage* location has a single start location. Table 3.4 shows Constraint 3.9 instantiated for the camera model.

T=0 Probabilistic Marking:

This constraint encodes the initial state probabilistic marking P_Θ of start locations Θ. It states that if a location is marked, then its *Start* must be enabled, and its state constraint must be consistent with observations. This constraint is formulated as:

$$\begin{aligned}&\forall C \in \Sigma_C, \forall L \in \Theta(C) : L^0 = Marked \Rightarrow \\ &(\mathit{Start}_L^0 = \mathit{Enabled} \wedge \mathit{State}_L^0 = \mathit{Consistent})\end{aligned} \tag{3.10}$$

Note that Constraint 3.10 is an implication rather than equivalence, since a location may be unmarked probabilistically even if its state constraint is consistent and its *Start* is enabled.

Constraint 3.10 is a soft constraint, in which tuples are mapped to a full range of probability values $\in [0..1]$. Each model M of Constraint 3.10, with *Scope*(M) $=\{L^0, \mathit{Start}_L^0, \mathit{State}_L^0\}$, is mapped to a probability value using the valuation function F_0. This captures the probabilistic marking of start locations. Essentially, if the state constraint of a start location L is consistent, then L is marked with probability $P_\Theta(L)$, and unmarked with probability $1 - P_\Theta(L)$:

$$F_0(M) = \begin{cases} \mathit{Prob}(L^0 = Marked) & if\ Condition1 \\ 1 - \mathit{Prob}(L^0 = Marked) & if\ Condition2 \\ 1.0 & otherwise \end{cases} \tag{3.11}$$

where $Condition1$ is:

$$(L^0 = Marked) \wedge (\mathit{Start}_L^0 = \mathit{Enabled}) \wedge (\mathit{State}_L^0 = \mathit{Consistent}) \tag{3.12}$$

and $Condition2$ is:

$$(L^0 = Unmarked) \wedge (\mathit{Start}_L^0 = \mathit{Enabled}) \wedge (\mathit{State}_L^0 = \mathit{Consistent}) \tag{3.13}$$

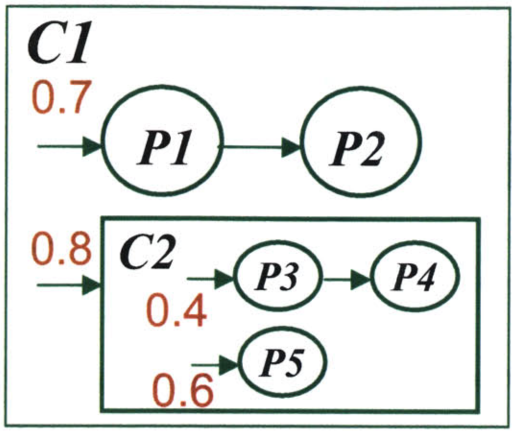

Figure 3-2: PHCA example with initial start probabilities.

As an example, consider the PHCA in Figure 3-2 which shows the initial probabilities of all start states (P_Θ in Definition 2-1). Constraint 3.10 for this PHCA is instantiated for each of the start locations *P1, C2, P3* and *P4*. Table 3.5 shows this instantiation for location $C2$.

3.1.6 Transition Constraints

Transition constraints encode the necessary conditions for enabling transitions, choosing probabilistically among consistent transitions, as well as for identifying and marking transition targets. PHCA transitions have the following properties, which are captured by the transition constraints:

First, PHCA have probabilistic, guarded transitions that originate from primitive

$C2^0$	$Start^0_{C2}$	$State^0_{C2}$	**Probability**
Marked	Enabled	Consistent	0.8
Unmarked	Enabled	Consistent	0.2
Unmarked	Disabled	Consistent	1.0
Unmarked	*	Inconsistent	1.0

Table 3.5: T=0 Probabilistic Marking example. The * notation indicates the full domain of the variable; in this case {*Enabled, Disabled*}

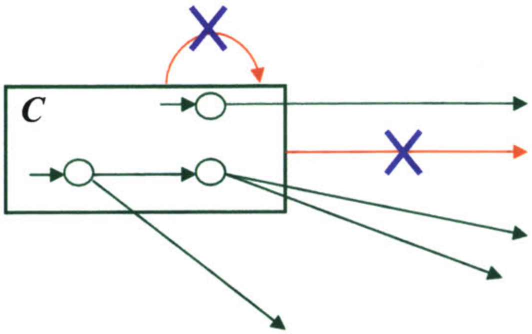

Figure 3-3: Transitions disallowed from composite PHCA locations.

locations, as specified by P_T in Definition 2-1. For example, in the camera model, there is a guarded transition from primitive location *Initializing* with guard *TakePicture*. It transitions to *TakingPicture* with probability 0.95, and transitions to *Broken* with probability 0.05. Note that the dotted transition from the *On* location is a shorthand for transitions from each of *On*'s subautomata to *Broken*. The general form of a PHCA transition is a tuple $(Source, Guard, Targets, Probability)$. *True* is considered to be a special case of a transition guard that is always satisfied. Recall that the consistency of the guards was encoded through Constraint 3.6.

Second, PHCA transitions are not allowed to originate at composite locations. This restriction is imposed in order to offer a simpler semantics. However, disallowing transitions from composite locations is not limiting, since a transition from a composite location can always be encoded as a set of transitions from primitive locations that are within the composite location. This is illustrated in Figure 3-3.

Third, each PHCA transition may have multiple targets, as illustrated by Figure 3-4. This property is in contrast to previous models, such as probabilistic concurrent constraint automata [53]. Multiple targets offer an advantage for modeling complex behavior, as discussed earlier.

Consider the PHCA in Figure 3-4. Multiple targets are marked simultaneously to model nested concurrent behaviors (such as *targets2*). Thus, for instance, if T2 is enabled (with *probability2*), then all locations in *targets2* must be marked. Likewise, if T1 is enabled (with *probability1*), its locations within *targets1* must be marked. If the

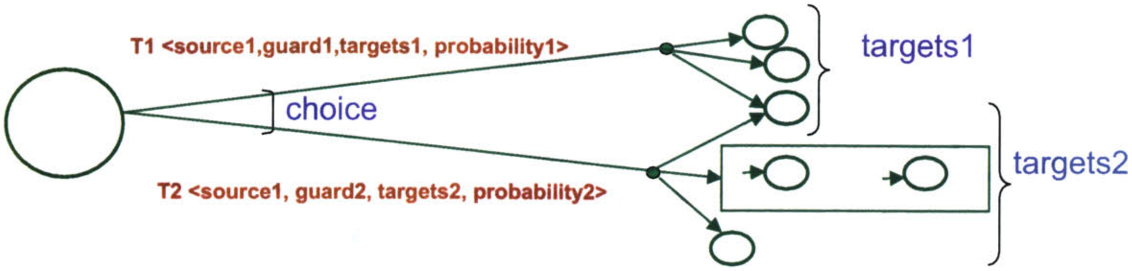
T1 <source1,guard1,targets1, probability1>
choice
T2 <source1, guard2, targets2, probability2>
targets1
targets2

Figure 3-4: PHCA transitions with multiple targets.

state constraint of a target is violated, the transition can not be enabled. Therefore, the semantics of marking multiple, possibly nested targets is modeled as a logical AND, representing conjunction of marked targets. This is captured by the transition target marking constraints, as well as the full marking constraints, presented in the next section.

Finally, making a probabilistic choice among transitions that originate from a primitive source location corresponds to choosing exactly one consistent transition from that source. Therefore, for each primitive location, probabilistic choice is encoded as an exclusive OR (XOR) among the location's outgoing transitions. Note that nested choice constructs may be used in the RMPL specification of the models, as described in [52]. In this case, the choice constructs are mapped into a probabilistic AND-OR tree, which is transformed using distribution into a two-level tree. The root node of the two-level tree represents probabilistic choice among a set of transitions, and the leaves represent the targets of each transition branch. The net effect is that a location can have multiple orthogonal sets of mutually exclusive transitions. This compilation step is presented in further detail in [52].

Figure 3-5 shows the logical mapping of transition choice, target marking, and target full marking, reflecting the semantics in [52]. The following starts by introducing the probabilistic choice constraints.

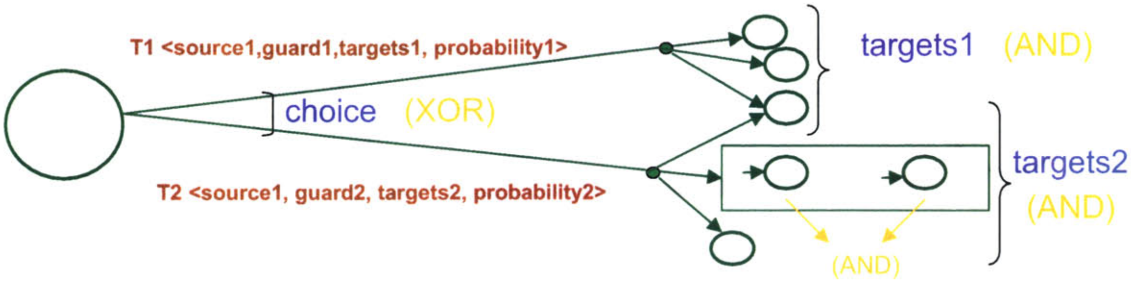
T1 <source1,guard1,targets1, probability1>
choice (XOR)
T2 <source1, guard2, targets2, probability2>
targets1 (AND)
targets2
(AND)
(AND)

Figure 3-5: Semantics of transition choice, target marking, and composite full marking.

Probabilistic Transition Choice:

This constraint encodes the probabilistic choice among transitions from each primitive location as an exclusive OR constraint, given by:

$$\begin{aligned}
&\forall t \in \{0..N-1\}, \forall P \in \Sigma_P : (\exists \tau \in T : \mathit{Source}(\tau) = P \Rightarrow \\
&[P^t = \mathit{Marked} \Leftrightarrow (\exists T \in \{T | \mathit{Source}(T) = P\} : T^t = \mathit{Enabled} \\
&\wedge (\forall T' \in (\{T | \mathit{Source}(T) = P\} - \{T\}) : T'^t = \mathit{Disabled}))] \bigwedge \\
&[P^t = \mathit{Unmarked} \Leftrightarrow (\forall T \in \{T | \mathit{Source}(T) = P\} : T^t = \mathit{Disabled})]) \quad (3.14)
\end{aligned}$$

This constraint specifies that a single transition is enabled among the possible outgoing transitions of a primitive location P, if and only if the P is marked.

Each model M of this constraint, with $\mathit{Scope}(M) = \{P^t\} \cup \{T_i^t | \mathit{Source}(T_i) = P\}$, is mapped to a probability value using the function:

$$F_T(M) = \begin{cases} \mathit{Prob}(T_i) & if\ (\exists T_i^t : T_i^t = \mathit{Enabled}) \\ 1.0 & otherwise \end{cases} \quad (3.15)$$

The probability of each tuple corresponds to the probability of the enabled transition that appears in that tuple.

The following example in Figure 3-6 shows a probabilistic choice between two transitions for a section of the PHCA in Figure 2-5. In order to encode this probabilistic choice, a location variable Off^t, with domain {Marked, Unmarked}, is

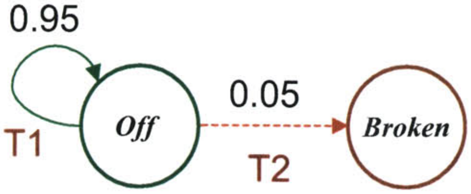

Figure 3-6: PHCA with two probabilistic transitions.

Off^t	$T1^t$	$T2^t$	**Probability**
Marked	Enabled	Disabled	0.95
Marked	Disabled	Enabled	0.05
Unmarked	Disabled	Disabled	1.0

Table 3.6: Probabilistic transition choice constraint.

first introduced for time t. Then, auxiliary variables $T1^t$ and $T2^t$, with domain $\{Enabled, Disabled\}$, are introduced for transitions T1 and T2, respectively.

The probabilistic transition choice constraint is instantiated at time t for the choice among the two transitions T1 and T2:

$$\begin{aligned} &Off^t = Marked \Leftrightarrow ((\exists T \in \{T1, T2\} : T^t = Enabled) \\ &\wedge(\forall T' \in (\{T1, T2\} - \{T\}) : T'^t = Disabled)) \bigwedge \\ &Off^t = Unmarked \Leftrightarrow (\forall T \in \{T1, T2\} : T^t = Disabled) \end{aligned} \tag{3.16}$$

This logical formula is compiled into a set of allowed tuples M, with associated probability values obtained using the function $F_T(M)$ above, as shown in Table 3.6.

Probabilistic choice is made among consistent transitions, as encoded by the following constraint.

Transition Consistency:

This constraint encodes the necessary, but not sufficient conditions for taking a consistent transition. This includes marking the transition's source location, and achieving consistency of the transition's guard condition, given the current state marking and

τ^t	$Initializing^t$	$Guard^t_\tau$	**Probability**
Enabled	Marked	Consistent	1.0
Disabled	*	*	1.0

Table 3.7: Transition constraint for the $(Initializing \rightarrow TakingPicture)$ transition, represented by variable τ at time t. Each * represents the full domain of the variable in the corresponding column.

issued commands.

$$\forall t \in \{0..N-1\}, \forall \tau \in T : \tau^t = Enabled \Rightarrow$$
$$(Source(\tau)^t = Marked \wedge Guard^t_\tau = Consistent) \tag{3.17}$$

Table 3.7 lists the allowed tuples of Constraint 3.17 for the transition $(Initializing \rightarrow TakingPicture)$. Similar constraints are generated for each PHCA transition.

The transition constraint in Formula 3.17, along with the guard consistency and transition choice constraints, specify that probabilistic choice is made among consistent transitions from marked primitive locations.

So far, the above constraints have not encoded the marking of transition targets, and in particular the full marking of composite targets. These are encoded through the constraints presented in the following sections. The next section first motivates the encoding of target marking constraints, and discusses the different categories of PHCA target locations that must be handled by these constraints.

Direct and Indirect Targets

As illustrated in Figure 3-4, there are two kinds of transition targets. The first type is a *direct target*, such as those in the set *targets1* of Figure 3-4. The second type is an *indirect target*, such as the start locations that will be marked through the full marking of their composite parent.

The distinction between direct and indirect targets is made because a direct target is marked by an enabled transition, while an indirect target is marked by its composite parent (by full marking). The full marking constraint must assure recursive propagation of a full marking through nested parallel processes (start locations) that are marked.

Consider the explicit marking of direct transition targets by the following formula:

$$\begin{aligned} &\forall t \in \{0..N-1\}, \forall \tau \in T: \\ &\tau^t = \mathit{Enabled} \Rightarrow (\forall L \in \mathit{Targets}(\tau) : L^{(t+1)} = \mathit{Marked}) \end{aligned} \tag{3.18}$$

While this constraint enforces marking, it does not enforce a target to be unmarked if there is no enabled transition to it. Therefore, rather than using Constraint 3.18 above, a set of constraints is introduced for specifying the marking and unmarking of each PHCA location. This set of constraints is referred to as **Marking Constraints**, to be discussed in Section 3.1.7. Marking constraints take into consideration:

1. The state constraint consistency of transition targets with commands and observations, as encoded by Formula 3.5. This consideration is important because a target with inconsistent state constraint can not be marked, and a transition to an inconsistent target can not be enabled.

2. Whether a direct transition to a target is enabled. The identification of direct transition targets is encoded by the Target Identification constraint presented below.

3. Whether the target has an enabled *Start*. That is, whether the target is marked by its composite parent. This is encoded by the Target Full Marking constraint presented below.

Refer to Figure 3-7 for an example of nested start locations that are indirect targets. If a transition to location $C1$ is enabled, $C1$ is identified as a direct transition target (Target Identification), and Target Full Marking of $C1$ enables the *Start* of

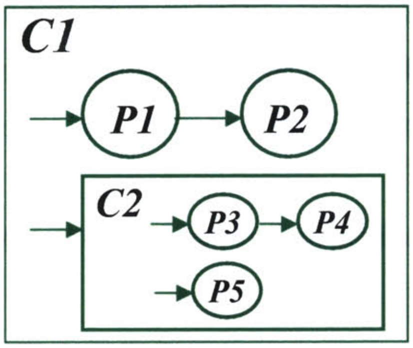

Figure 3-7: Nested start locations; each start location is indicated by a small arrow.

locations *P1* and $C2$. Once the *Start* of location $C2$ is enabled, $C2$ is considered an indirect target; therefore, the Target Full Marking constraint recursively enables the *Start* of *P3* and *P5*. Once Target Identification and Target Full Marking identify direct targets and enable the *Start* of indirect targets, the Marking Constraints use this information to encode the marking and unmarking of each location, based on the considerations listed above.

The formulation of the Target Identification and Target Full Marking constraints is presented below, followed by the formulation of Marking Constraints in the Section 3.1.7.

Target Identification:

The following constraint enforces that a target may be enabled (transitioned to) if and only if at least one of the transitions to the target is enabled. This is formulated as:

$$\forall t \in \{0..N-1\}, \forall L \in \Sigma : TransitionTo_L^{(t+1)} = \mathit{Enabled} \Leftrightarrow$$
$$(\exists \tau \in \{T|Target(T) = L\} : \tau^t = \mathit{Enabled}) \qquad (3.19)$$

Note that $TransitionTo_L$ is a variable introduced for each PHCA location that is a direct transition target, as illustrated in Figure 3-10.

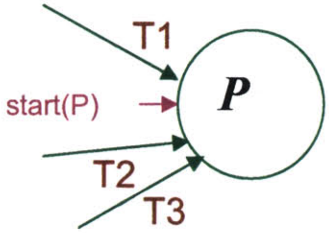

Figure 3-8: Target Identification example. P is a primitive start location; T1, T2 and T3 are transitions; start(P) indicates that P is a start location.

For example, consider the primitive location P in Figure 3-8. The instantiation of Constraint 3.19 generates the mutually exclusive tuples in Table 3.8. $T1, T2$ and $T3$ are variables introduced for each of the incoming transitions of P.

Note that the target identification constraint is instantiated only for PHCA locations that have incoming transitions, and thus are direct targets. The Target Full Marking constraint, presented in the next section, incorporates the identification of indirect targets. Indirect targets are start locations that are enabled through the full marking semantics discussed in Section 3.1.3.

Target Full Marking:

This constraint captures the full marking semantics of composite targets, by recursively enabling the *Start* of each of the composite location's subautomata. Unlike the T=0 Full Marking constraint in Formula 3.9, for $t > 0$ the marking of a composite location C does not enable the *Start* of its subautomata. Otherwise, all the start locations of C would be enabled as long as C is marked. Therefore, unlike T=0 full

$TransitionTo_P^{(t+1)}$	$T1^t$	$T2^t$	$T3^t$	**Probability**
Enabled	Enabled	*	*	1.0
Enabled	Disabled	Enabled	*	1.0
Enabled	Disabled	Disabled	Enabled	1.0
Disabled	Disabled	Disabled	Disabled	1.0

Table 3.8: Target identification constraint. Each * represents the full domain of the variable in the corresponding column.

$TransitionTo^t_{C1}$	$Start^t_{C1}$	$Start^t_{P1}$	$Start^t_{C2}$	**Probability**
Enabled	*	Enabled	Enabled	1.0
Disabled	Enabled	Enabled	Enabled	1.0
Disabled	Disabled	Disabled	Disabled	1.0

$Start^t_{C2}$	$Start^t_{P4}$	$Start^t_{P5}$	**Probability**
Enabled	Enabled	Enabled	1.0
Disabled	Disabled	Disabled	1.0

Table 3.9: Full marking of $C1$ and $C2$ from Figure 3-7. Each * represents the full domain $\{$*Enabled, Disabled*$\}$.

marking, target full marking encodes the following. The start locations of a composite location C are enabled if and only if a transition to C is enabled, or if C is an enabled start location. Furthermore, the start locations of C are disabled if and only if all transitions to C are disabled, and C is not an enabled start location. This formulation is given by:

$$\begin{aligned}
&\forall t \in \{1..N\}, \forall C \in \Sigma_C :\\
&[(\forall L \in \Theta(C) : Start^t_L = \mathit{Enabled}) \Leftrightarrow\\
&(TransitionTo^t_C = \mathit{Enabled} \vee Start^t_C = \mathit{Enabled})] \bigwedge\\
&[(\forall L \in \Theta(C) : Start^t_L = \mathit{Disabled}) \Leftrightarrow\\
&(TransitionTo^t_C = \mathit{Disabled} \wedge Start^t_C = \mathit{Disabled})]
\end{aligned} \tag{3.20}$$

As illustrated by the scenario in Figure 3-7, full marking is applied to both direct composite targets and composites with enabled *Start* (that is, they are enabled start locations). This is captured in Formula 3.20 through assignments to the $TransitionTo_C$ and $Start_C$ variables.

For example, Tables 3.9 list the constraint tuples generated for the PHCA in Figure 3-7, for each of the $C1$ and $C2$ composite locations. $C1$ is assumed to be both a direct target and a start location. On the other hand, $C2$ is just a start location (an indirect target).

Note that for the special case of an indirect target location, a *TransitionTo* variable is not generated. Location $C2$ in the Table 3.9 example demonstrates this case. The assignment $TransitionTo = Disabled$ can, therefore, be substituted by *true* to simplify Constraint 3.20 to:

$$\begin{aligned}
&\forall t \in \{1..N\}, \forall C \in \Sigma_C : \\
&[(\forall L \in \Theta(C) : Start_L^t = Enabled) \Leftrightarrow Start_C^t = Enabled] \bigwedge \\
&[(\forall L \in \Theta(C) : Start_L^t = Disabled) \Leftrightarrow Start_C^t = Disabled]
\end{aligned} \tag{3.21}$$

Building upon the Target Identification and Target Full Marking constraints, the following section presents the constraints for marking the target locations.

3.1.7 Marking Constraints

The above constraints identify the enabled transitions and enforce the full marking of targets; however, they do not encode the process of taking transitions. The target marking constraints presented below encode this process based on the state constraint consistency of a target, for both direct targets and start locations.

The marking constraints are instantiated for each location in a PHCA. The following sections present three types of marking constraints: Primitive Target Marking, Composite Target Marking, and Hierarchical Composite Marking/Unmarking (recall Section 3.1.3). Each of these constraints is formulated below.

Primitive Target Marking:

This constraint formulates the marking and unmarking of each primitive location, based on whether its state constraint is consistent, and whether it is a direct target or an indirect target. Recall that an indirect target is a start location of a composite.

The primitive target marking constraint encodes that a primitive location can be marked if and only if: 1) its state constraint is consistent with commands and observations AND 2) either a transition to it is enabled OR its *Start* is enabled by full marking of its composite parent. Otherwise, the location is unmarked.

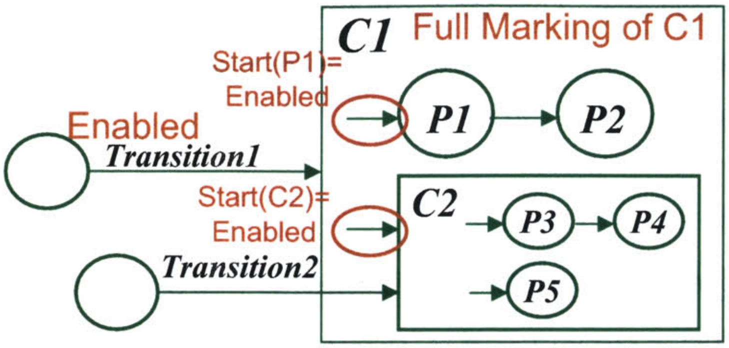

Figure 3-9: Marking example scenario.

This constraint is formulated as:

$$\forall t \in \{1..N\}, \forall P \in \Sigma_P : [P^t = Marked \Leftrightarrow$$
$$(TransitionTo_P^t = Enabled \vee Start_P^t = Enabled) \wedge State_P^t = Consistent]$$
$$\bigwedge [P^t = Unmarked \Leftrightarrow (TransitionTo_P^t = Disabled \wedge Start_P^t = Disabled)] \tag{3.22}$$

The case $t = 0$ is not included in Constraint 3.22, because targets do not exist in the initial state (t=0). However, the marking of locations at time $t = 0$ is encoded by the T=0 Constraints presented previously in Section 3.1.5.

To illustrate primitive target marking, consider the scenario in Figure 3-9: Transition1 is enabled, thereby enabling the full marking of C1. This means that, in addition to identifying C1 as a direct transition target (see Target Identification constraint), the *Start* variables to locations *P1* and $C2$ will be enabled by the Target Full Marking constraint. Suppose that it is possible to mark $C1$, because its state constraint is consistent; then, *P1* will be marked via the Primitive Marking constraint.

Location *P1* in Figure 3-9 is the special case of an indirect target with start location. Therefore, Constraint 3.27 is instantiated to generate the allowed tuples listed in Table 3.10.

$P1^t$	$Start^t_{P1}$	$State^t_{P1}$	**Probability**
Marked	Enabled	Consistent	1.0
Unmarked	Disabled	*	1.0

Table 3.10: Marking of primitive location *P1* in Figure 3-9.

Composite Target Marking:

This constraint formulates the marking and unmarking of each composite location. Similar to the primitive case, marking of a composite location depends on the consistency of its state constraint with commands and observations, and whether it is a direct target or a start location. However, unlike the primitive case, a composite location has the added requirement that it must be marked when one of its subautomata is marked. The composite marking constraint is formulated as:

$$\begin{aligned}
&\forall t \in \{1..N\}, \forall C \in \Sigma_C : [C^t = Marked \Rightarrow State^t_C = Consistent] \\
&\bigwedge [C^t = Marked \Leftarrow (TransitionTo^t_C = Enabled \vee Start^t_C = Enabled) \\
&\wedge State^t_C = Consistent] \\
&\bigwedge [C^t = Unmarked \Rightarrow (TransitionTo^t_C = Disabled \\
&\wedge Start^t_C = Disabled)]
\end{aligned} \tag{3.23}$$

This is a "weaker" version of the primitive target marking constraint (Formula 3.22), which ensures consistency with the hierarchical composite marking/unmarking constraint presented in the next section. The weakening of Constraint 3.23 is achieved by using implication, as opposed to the equivalence in the primitive marking constraint 3.22.

As an example of composite marking, consider the PHCA in Figure 3-9. The composite marking constraint is instantiated for each of locations $C1$ and $C2$; Table 3.11 lists the tuples that are models of each constraint instance. Location $C1$ is a direct transition target. On the other hand, $C2$ is both a transition target and a start location.

$C1^t$	$TransitionTo^t_{C1}$	$State^t_{C1}$	**Probability**
Marked	*	Consistent	1.0
Unmarked	Disabled	*	1.0

$C2^t$	$TransitionTo^t_{C2}$	$Start^t_{C2}$	$State^t_{C2}$	**Probability**
Marked	*	*	Consistent	1.0
Unmarked	Disabled	Disabled	*	1.0

Table 3.11: Composite marking of the $C1$ and $C2$ locations in Figure 3-9. Each * represents the full domain of the variable in the corresponding column.

Consider the particular scenario in Figure 3-9, in which Transition1 is enabled. This corresponds to $TransitionTo_{C1}$ = *Enabled* in Table 3.11, which is consistent with the tuple in the first row of $C1$'s constraint. If, furthermore, the state constraint of $C1$ is consistent, the tuple is consistent and thus the constraint is satisfied. On the other hand, if the state constraint of $C1$ is not consistent with some observations or commands, then in order for the constraint to be satisfiable, Transition1 must be disabled. This demonstrates that the constraint will enforce the enabling of those transitions for which the targets can be marked.

The composite marking constraint alone may not necessarily determine the marking and unmarking of composites. For example, if both *TransitionTo* and *Start* are disabled and *State* is consistent, the first two tuples of $C2$'s constraint in Table 3.11 will be satisfied. The following section introduces the constraint for incorporating the hierarchical semantics of PHCA, which will further restrict the composite marking.

Hierarchical Composite Marking/Unmarking:

This constraint encodes how the marking and unmarking of each composite location is affected by the state of its subautomata, as discussed in Section 3.1.3. This constraint complements the full marking constraints that encode how marking of a composite location affects the marking of its subautomata (Constraints 3.9 and 3.20).

Hierarchical marking/unmarking is formulated as:

$$\begin{aligned}
&\forall t \in \{0..N\}, \forall C \in \Sigma_C : \\
&[C^t = \mathit{Marked} \Leftrightarrow (\exists L \in \mathit{Subautomata}(C) : L^t = \mathit{Marked})] \bigwedge \\
&[C^t = \mathit{Unmarked} \Leftrightarrow (\forall L \in \mathit{Subautomata}(C) : L^t = \mathit{Unmarked})] \qquad (3.24)
\end{aligned}$$

Subautomata(C) is the set of locations that are the children of C. Constraint 3.24 specifies that a composite location is unmarked if and only if all of its subautomata are unmarked; furthermore, a composite location is marked if and only if at least one of its subautomata is marked.

Referring back to the example scenario in Figure 3-9, Constraint 3.24 is instantiated for each of locations $C1$ and $C2$. Table 3.12 shows the mutually exclusive set of tuples for each instantiation of the constraint.

Recall from the discussion in the previous section that the first two tuples of the composite marking constraint for $C2$ in Table 3.11 are satisfied in case *TransitionTo* and *Start* are disabled and *State* is consistent. Then, the hierarchical constraint for $C2$, shown in Table 3.12, enforces the marking or unmarking of $C2$ based on the marking of its subautomata.

$C1^t$	$P1^t$	$P2^t$	$C2^t$	**Probability**
Marked	Marked	*	*	1.0
Marked	Unmarked	Marked	*	1.0
Marked	Unmarked	Unmarked	Marked	1.0
Unmarked	Unmarked	Unmarked	Unmarked	1.0

$C2^t$	$P3^t$	$P4^t$	$P5^t$	**Probability**
Marked	Marked	*	*	1.0
Marked	Unmarked	Marked	*	1.0
Marked	Unmarked	Unmarked	Marked	1.0
Unmarked	Unmarked	Unmarked	Unmarked	1.0

Table 3.12: Hierarchical composite marking/unmarking of the $C1$ and $C2$ locations in Figure 3-9. *Subautomata*$(C1) = \{P1,P2,C2\}$ and *Subautomata*$(C2) = \{P3,P4,P5\}$. Each * represents the full domain of the variable in the corresponding column.

3.1.8 Special Cases of Marking Constraints

To develop a compact encoding for the Marking Constraints, PHCA locations are divided into four categories. Figure 3-10 shows these categories for both primitive and composite locations, in the left and right columns, respectively. In the first case, a location is both a direct target of one or more transitions, and a start location of a composite automaton. This case is shown in Figure 3-10(a) for a primitive location P and a composite location C. Figures 3-10(b) and 3-10(c) show special cases of 3-10(a) in which the location is a direct target but not a start, and only a start but not a direct target, respectively. Finally, Figure 3-10(d) is the case in which a location is neither a direct target nor a start location. This last case does not hold for primitive locations; however, a composite location of this category is possible, since it can get marked through the hierarchical composite marking/unmarking semantics discussed in Section 3.1.3.

The primitive and composite target marking constraints, formulated above, can be instantiated for all categories of locations shown in Figure 3-10. However, more compact encodings of the primitive and composite marking constraints can be obtained for the special cases of locations in Figure 3-10 (b), (c), (d). The following sections begin by discussing the two special cases of the primitive target marking constraint specified in Formula 3.22.

Primitive Marking Case 1: direct target, non-start location This case is illustrated in the left column of Figure 3-10(b). The variable *Start* is not generated for this case of non-start location. Constraint 3.22 is then simplified by substituting $Start_P = \mathit{Disabled}$ with *true* and similarly $Start_P = \mathit{Enabled}$ with *false*:

$$\begin{aligned}&\forall t \in \{1..N\}, \forall P \in \Sigma_P : [P^t = Marked \Leftrightarrow \\ &(TransitionTo^t_P = \mathit{Enabled} \wedge State^t_P = Consistent)] \\ &\bigwedge [P^t = Unmarked \Leftrightarrow (TransitionTo^t_P = \mathit{Disabled})]\end{aligned} \qquad (3.25)$$

This captures the desired semantics that targets of enabled transition must be marked

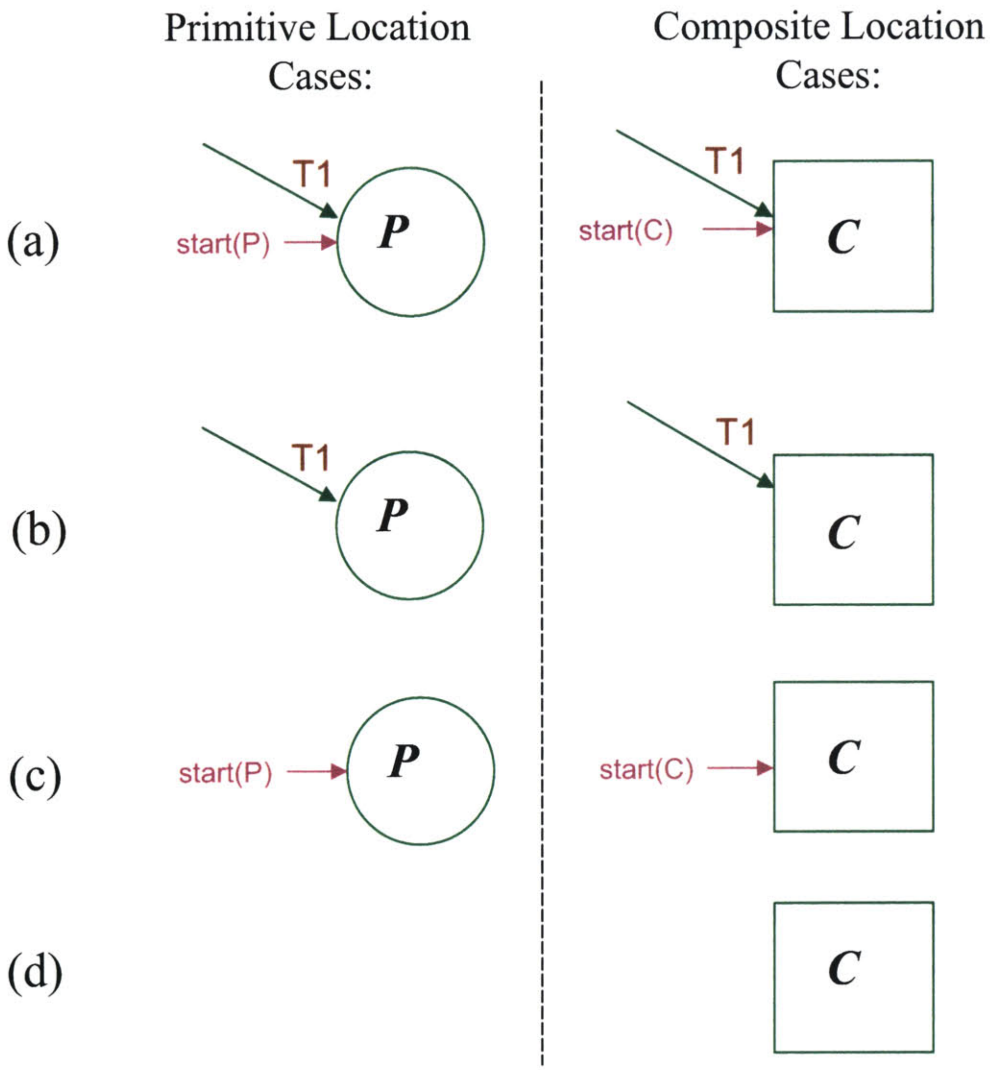

Figure 3-10: Categories of PHCA locations: (a) direct target and start location; (b) direct target and non-start location; (c) indirect target and start location; (d) indirect target and non-start location. Category (d) is possible only for composite locations. P refers to a primitive location; C refers to a composite location; T1 refers to a transition; start refers to a start location.

(AND constraint), otherwise the transition can not be enabled. To see this, Constraint 3.25 can be broken up into the following two formulae:

$$\begin{cases} \forall t \in \{1..N\}, \forall P \in \Sigma_P : P^t = Marked \Leftrightarrow TransitionTo_P^t = Enabled \\ \forall t \in \{1..N\}, \forall P \in \Sigma_P : P^t = Marked \Rightarrow State_P^t = Consistent \end{cases} \tag{3.26}$$

Primitive Marking Case 2: indirect target, start location

This case is illustrated in the left column of Figure 3-10(c). The variable *TransitionTo* is not generated for this location, since it is not a direct target of a transition. Similar to special case 1, the Constraint 3.22 is simplified by substituting $TransitionTo_P = Disabled$ with *true* and $TransitionTo_P = Enabled$ with *false*:

$$\begin{aligned} & \forall t \in \{1..N\}, \forall P \in \Sigma_P : [P^t = Marked \Leftrightarrow \\ & (Start_P^t = Enabled \wedge State_P^t = Consistent)] \\ & \bigwedge [P^t = Unmarked \Leftrightarrow \\ & Start_P^t = Disabled] \end{aligned} \tag{3.27}$$

Note that this captures the semantics that all enabled start locations must be marked. This corresponds to the AND semantics for the marking of start locations, as shown in Figure 3-5.

The following sections discuss special cases of the composite target marking constraint, specified in Formula 3.23.

Composite Marking Case 1: direct target, non-start location

This case is illustrated in the right column of Figure 3-10(b). Similar to the primitive location case, the variable *Start* is not generated for a non-start composite location. Substituting $Start_C = Disabled$ with *true* and $Start_C = Enabled$ with *false* simplifies Constraint 3.23 to:

$$\forall t \in \{1..N\}, \forall C \in \Sigma_C : [C^t = Marked \Rightarrow State^t_C = Consistent]$$
$$\bigwedge [C^t = Marked \Leftarrow (TransitionTo^t_C = Enabled \wedge State^t_C = Consistent)]$$
$$\bigwedge [C^t = Unmarked \Rightarrow TransitionTo^t_C = Disabled] \tag{3.28}$$

Note that the third term

$$[C^t = Unmarked \Rightarrow TransitionTo^t_C = Disabled]$$

of Constraint 3.28 encodes unmarking of a composite location as an implication specifying that all transitions to it are disabled; this is equivalent to specifying that enabling a transition to the composite location implies the marking of that location. This effectively encodes the deterministic AND constraint for marking the targets of each enabled transition.

Composite Marking Case 2: indirect target, start location

This case is illustrated in the right column of Figure 3-10(c). The variable *TransitionTo* is not generated for this location, since it is not a direct target of a transition. Substituting $TransitionTo_C = Disabled$ with *true* and $TransitionTo_C = Enabled$ with *false* simplifies Constraint 3.23 to:

$$\forall t \in \{1..N\}, \forall C \in \Sigma_C : [C^t = Marked \Rightarrow State^t_C = Consistent]$$
$$\bigwedge [C^t = Marked \Leftarrow (Start^t_C = Enabled \wedge State^t_C = Consistent)]$$
$$\bigwedge [C^t = Unmarked \Rightarrow Start^t_C = Disabled] \tag{3.29}$$

The above constraint is analogous to the Constraint 3.28, which captures the case of direct transition targets (Figure 3-10(b)).

Composite Marking Case 3: indirect target, non-start location

This case is illustrated in the right column of Figure 3-10(d), and is only possible for a composite location. Neither the *TransitionTo* nor the *Start* variables are generated for this location, since it is neither a direct target nor a start location. The substitutions $(TransitionTo_C = Disabled) \equiv true$, $(TransitionTo_C = Enabled) \equiv false$, $(Start_C = Disabled) \equiv true$ and $(Start_C = Enabled) \equiv false$ simplify Constraint 3.23 to:

$$\forall t \in \{1..N\}, \forall C \in \Sigma_C : C^t = Marked \Rightarrow State_C^t = Consistent \qquad (3.30)$$

3.1.9 Summary of Constraint Encoding

The above sections have presented the encoding of PHCA as a soft constraint optimization problem (COP). For convenience, the constraints are also listed in Appendix A. The encoding forms the basis for solving the N-Stage best-first trajectory enumeration (N-BFTE) problem for PHCA. The COP formulation includes consistency conditions within the PHCA, and initial probabilistic markings of the PHCA, as well as the transition and marking constraints that capture the semantics of the PHCA models. Given a command sequence, solving the COP will simulate the PHCA models over several time steps, along with checking consistency with observations. Recall that the COP formulation is performed offline, when observations and commands are not yet available. Consistency checking necessitates the addition of constraints that encode the assignments to observable and control variables. Therefore, the COP is dynamically updated in the online phase as the time horizon shifts, and observations become available. The updates to the COP are presented in Chapter 4, as part of the online N-BFTE algorithm.

Recall from Figure 1-2 that, subsequent to the COP formulation in the offline phase, the COP is decomposed into independent subproblems, in order to enhance the efficiency of the online solution phase. This decomposition step is discussed in the following section.

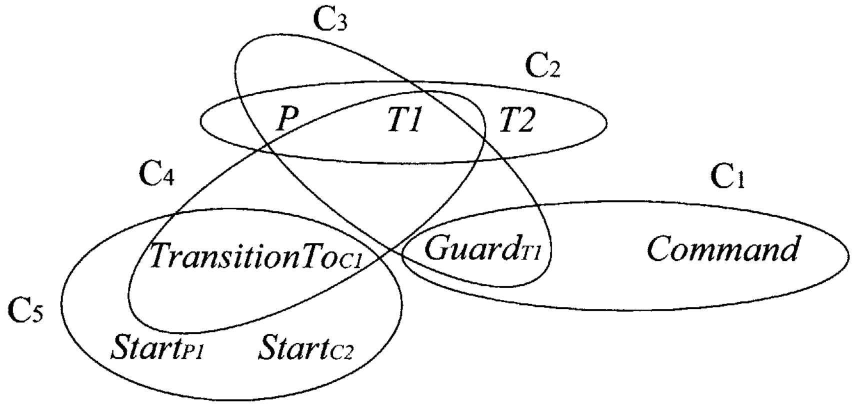

Figure 3-11: Hypergraph of a constraint network.

3.2 Tree Decomposition of the COP

The COP formulated in the previous sections forms a constraint network (X, D, C), where C is a set of soft (valued) constraints. A constraint network can be represented as a hypergraph, which is defined as follows [17].

Definition 3-3 (Hypergraph)
A hypergraph $H = (V, S)$ is a structure that consists of a set of vertices $V = \{v_1, ..., v_n\}$ and a set of subsets of these vertices $S = \{S_1, ..., S_m\}$, where $S_i \subset V$, called hyperedges. As opposed to a regular edge, a hyperedge can connect more than two variables.

The structure of a COP (X, D, C) can be represented as the hypergraph $H = (X, S)$, where X is the set of COP variables, and S is the set of scopes of the constraints in C.

Figure 3-11 shows an example of a hypergraph for a COP based on some of the constraints presented in the previous section. The vertices of the hypergraph correspond to the variables $\{P$, $T1$, $T2$, $Guard_{T1}$, $Command$, $TransitionTo_{C1}$, $Start_{P1}$, $Start_{C2}\}$ and the hyperedges are the scopes of the constraints $\{C_1, C_2, C_3, C_4, C_5\}$. Each C_i is an instance of a constraint formulated in the previous sections: C_1 is an instance of the transition guard consistency constraint in Formula 3.6, with scope

$\{Guard_{T1},\ Command\}$; C_2 is an instance of the probabilistic transition choice constraint in Formula 3.14, with scope $\{P,\ T1,\ T2\}$; C_3 is an instance of the transition consistency constraint in Formula 3.17, with scope $\{T1,\ P,\ Guard_{T1}\}$; C_4 is an instance of the target identification constraint in Formula 3.19, with scope $\{T1,\ TransitionTo_{C1}\}$, and C_5 is an instance of the target full marking constraint in Formula 3.21, with scope $\{TransitionTo_{C1},\ Start_{P1},\ Start_{C2}\}$.

A hypergraph reflects the acyclic structure of the constraints in the COP. By exploiting the structure of the constraints, the COP can be transformed into an equivalent, acyclic instance. Such a transformation enhances the efficiency of solving the COP, through the application of techniques like dynamic programming [17]. An acyclic form of the COP can be obtained through tree decomposition [24, 31] of the COP:

Definition 3-4 (Tree Decomposition)

A *tree decomposition* for a COP (X, D, C) with variables X, domains D and valued constraints C is a triple (T, χ, λ), where $T = (N, E)$ is a rooted tree with nodes N and edges E, and the labeling functions $\chi(n_i) \subseteq X$, and $\lambda(n_i) \subseteq C$ are defined such that:

1. For each constraint $c_j \in C$, there exists exactly one node $n_i \in N$ such that $c_j \in \lambda(n_i)$. For this n_i, $\mathrm{var}(c_j) \subseteq \chi(n_i)$ (covering condition);
2. For each variable $x_i \in X$, the set $\{n_j \in N \mid x_i \in \chi(n_j)\}$ of nodes labeled with x_i induces a connected subtree of T (connectedness condition).

Figure 3-12 shows a tree decomposition of the COP hypergraph in Figure 3-11. The tree decomposition results in an acyclic COP (X, D, C'), where C' is obtained by composing the constraints in $\lambda(n_i)$:

$$C' = \bigcup_{n_i \in N} \left(\bigoplus_{C_i \in \lambda(n_i)} C_i \right) \tag{3.31}$$

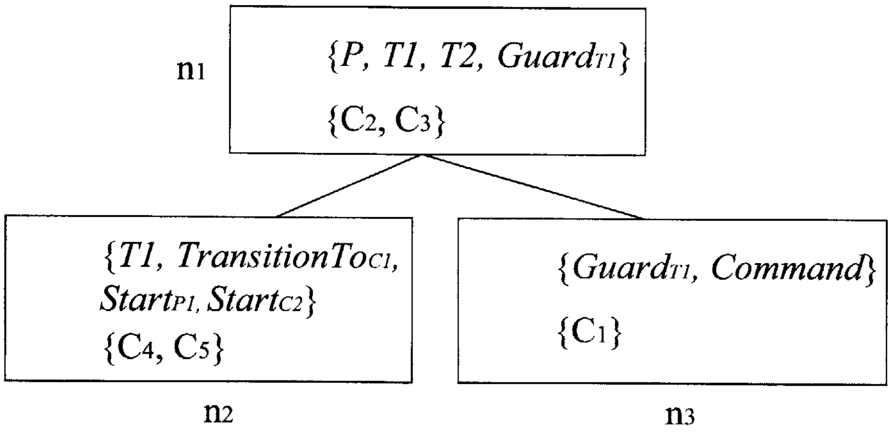

Figure 3-12: A tree decomposition of the hypergraph in Figure 3-11.

The operator $\bigoplus$ in Equation 3.31 (see Section 3.1.1) joins the constraints associated with each tree node, while multiplying the values of each pair of joined constraint tuples.

Recall that observations and commands are not available in the offline phase, when tree decomposition is applied to the COP. As highlighted in the previous section, the COP must, therefore, be dynamically updated in the online phase, in order to incorporate assignments to observable variables. However, updating the COP in the online phase does not affect the offline tree decomposition; since each observation is a *unary* constraint over an observable variable O_i, it can be added to the tree decomposition as a leaf child of any node n_i for which $O_i \in \chi(n_i)$ [47]. This means that the constraint is inserted into the tree as a leaf of any node that includes O_i in its set of variables. The implication of this is that the observations and commands can be incorporated into the tree structure during the online phase.

Tree decomposition is revisited briefly in Chapter 4, in the context of the online monitoring and diagnosis phase.

3.3 Summary of the Offline Phase

In summary, this chapter introduced the offline formulation of the N-Stage Best-First Trajectory Enumeration (N-BFTE) problem, as a soft COP that encodes the

PHCA models and their probabilistic execution semantics as valued constraints. The application of tree decomposition to the COP was discussed. The next chapter builds upon this offline formulation, by introducing the online PHCA-based monitoring and diagnosis phase.

Chapter 4

Online Phase: Best-First Trajectory Tracking

The previous chapter introduced the offline encoding of PHCA models and their execution semantics as a valued COP over a finite time horizon. The COP was subsequently decomposed into a tree structure. Building upon this offline formulation, this chapter presents the online monitoring and diagnosis process that enumerates and tracks the most likely PHCA state trajectories within the finite time horizon. Figure 1-2 illustrates the online phase, which incorporates observations and commands into the diagnosis process.

4.1 PHCA State Trajectories

Given a PHCA model, recall that N-BFTE is the problem of finding the most likely trajectories of system state that are consistent with a sequence of observations, commands, and the PHCA model within the N-Stage time horizon (Section 2.3).

In the offline phase, the N-BFTE problem was formulated as a valued COP that encodes the simulation of PHCA models given commands, as well as the conditions for consistency with observations within the time horizon. This encoding thus effectively restricts the trajectories of PHCA state evolution. The possible evolutions of PHCA states over the N-Stage horizon, as encoded by the valued constraints, can be represented as a *Trellis diagram*, shown in Figure 2-7(a). In the context of N-BFTE, the Trellis diagram is unfolded into a branching tree, as shown in Figure 2-7(b). The branches of this tree are the possible evolutions of PHCA state trajectories. Solving the N-BFTE problem thus corresponds to finding the most likely branches of this tree, as defined by Equation 2.1. Referring to the vision-based navigation scenario in Figure 2-4 represents an example of most likely PHCA state trajectories. The trajectories are found by updating and solving the tree-structured COP formulated in the previous chapter. This is accomplished through the online N-BFTE process, which is described in Section 4.2.

4.2 The N-BFTE Algorithm

The N-BFTE problem, formulated during the offline phase as a soft COP, is dynamically updated and solved during the online phase when new observations and commands become available. The steps of the online N-BFTE process are given by the pseudocode in Figure 4-1. First, the COP is updated as follows. New observations taken during runtime, as well as commands issued to the system are added to the COP. As the time horizon shifts, observations and commands are truncated accordingly. Furthermore, all initial constraints are removed from the COP. However, in order to track the trajectories from the previous horizon, a new constraint is added to the COP. This latter constraint restricts the set of initial states within the new horizon. Finally, the updated COP is solved using an optimal constraint solver, and the process repeats. The following discusses each step of the process in further detail.

Consider the initial time horizon $[0..N]$. During the first iteration of N-BFTE, observations and commands are added to the COP as unary constraints with probability 1 (part 1.1 of Figure 4-1). These constraints will be referred to as the history of observations $H_O^{(t \rightarrow t+N)}$ and the history of commands $H_C^{(t \rightarrow t+N-1)}$ within the N-Stage horizon. Each constraint within H_O and H_C represents an assignment to observable and command variables, respectively. For example, if a command *TakePicture* is issued to the camera module at time t, the unary constraint $command^t = TakePicture$

```
N-BFTE ($COP$, $Traj_{(i=1..K)}^{(t \rightarrow t+N)}$, $H_O^{(t \rightarrow t+N)}$, $H_C^{(t \rightarrow t+N-1)}$) $\rightarrow Traj_{(i=1..K)}^{(t+1 \rightarrow t+N+1)}$ ::

 1. Update the $COP$ $(X, D, V)$ to obtain $COP'$ $(X, D, V')$:

   1.1 If this is the first iteration of N-BFTE, then:
           $V' = V \cup H_O^{(t \rightarrow t+N)} \cup H_C^{(t \rightarrow t+N-1)}$.
       Skip to step 2.

   1.2 Shift the time horizon from $(t \rightarrow t+N)$ to $(t+1 \rightarrow t+N+1)$
       by deleting observations $O^{(t)}$ and commands $C^{(t)}$ from constraints
       $H_O$ and $H_C$.

   1.3 Add new observations $O^{(t+N+1)}$ and commands $C^{(t+N)}$ to the COP:
           $H_O^{(t+1 \rightarrow t+N+1)} = O^{(t+N+1)} \cup H_O^{(t+1 \rightarrow t+N)}$
           $H_C^{(t+1 \rightarrow t+N)} = C^{(t+N)} \cup H_C^{(t+1 \rightarrow t+N-1)}$

   1.4 Remove the T=0 Constraints if the horizon is shifted from $t = 0$,
       and remove the Initial State Constraint (see 1.5) if $t > 0$.

   1.5 To track trajectories $Traj_{(i=1..K)}^{(t \rightarrow t+N)}$, constrain the initial state assignments
       $L^{(t+1)}$ of $Traj_{(i=1..K)}^{(t+1 \rightarrow t+N+1)}$ to be consistent with $S_{(i=1..K)}^{(t+1)} \in Traj_{(i=1..K)}^{(t \rightarrow t+N)}$,
       where each state represents an assignment to solution variables $L_\Sigma$.
       Add to $V$ the Initial State Constraint:

           $(\forall\ L \in \Sigma : L^{(t+1)} \wedge (\bigvee_{i=1..K} S_i^{(t+1)}))$,

       with valuation function $F_I$ that maps each model $M$ of this constraint
       to a probability value given by:

           $F_I(M) = P(S_i^{'(t)}) \cdot P_T(S_i^{(t+1)} | S_i^{'(t)}, \Delta^{(t)})$,
               if $L^{(t+1)} \wedge S_i^{(t+1)}$ are consistent; 0 otherwise.

       The set of states $S_i^{'(t)})$ denote the beginning states of $Traj_{(i=1..K)}^{(t \rightarrow t+N)}$.
       The probabilities $P(S_i^{'(t)})$ are obtained using the valuation function
       $F_0$ if $t = 0$, or through previous iterations of N-BFTE otherwise.

 2. Enumerate and return the $K$ most likely trajectories $Traj_{(i=1..K)}^{(t+1 \rightarrow t+N+1)}$ by
    solving the COP for sets of solution variables $\{L_\Sigma^{(t+1)}, \ldots, L_\Sigma^{(t+1+N)}\}$, using
    an optimal constraint solver.

 3. Repeat.
```

Figure 4-1: Online N-BFTE process.

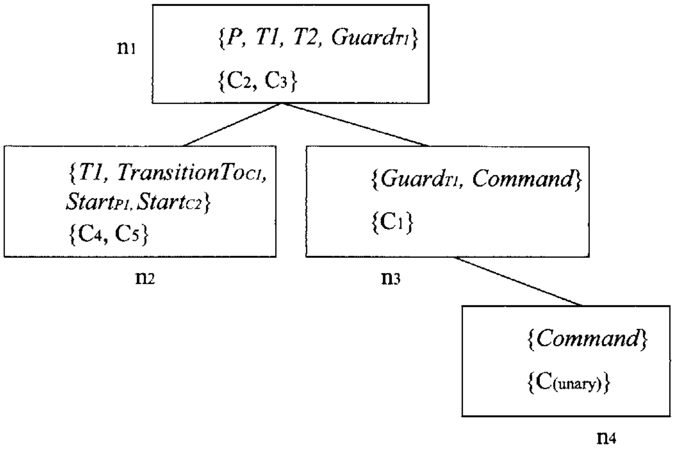

Figure 4-2: Addition of a unary constraint to the tree decomposition in Figure 3-12.

is added to the COP. This constraint generates a single tuple with value 1.

Recall that the COP generated in the offline phase was decomposed into a tree structure (Section 3.2). Adding unary constraints does not alter the structure of the COP, because the constraint can be added as a leaf node. This is illustrated in Figure 4-2 for the example presented in Section 3.2.

The COP is solved within the initial time horizon, after the addition of observations and commands. The solutions correspond to the K most likely trajectories, in decreasing order of likelihood, defined by Equation 2.1. Figure 4-3(a) illustrates two most likely trajectories found within the initial time horizon $[t_0..t_n]$. Recall that the branching tree represents evolutions of PHCA state trajectories.

As time progresses during the online solution phase, the N-stage horizon is shifted from $[t..t+N]$ to $[t+1..t+N+1]$. The history of observations and commands outside the N-Stage horizon is truncated (part 1.2 of Figure 4-1). Deleting an observation corresponds to setting the tuples of its unary constraint to all possible values allowed by the domain of the observable variable. The commands are treated similarly. Furthermore, shifting the time horizon requires the addition of any new observations and issued commands (part 1.3 of Figure 4-1). This is accomplished as described above, by adding a new unary constraint to the tree structure, if it does not exist already for that observable variable, and by setting the tuple of the constraint to the observed value, with probability 1.

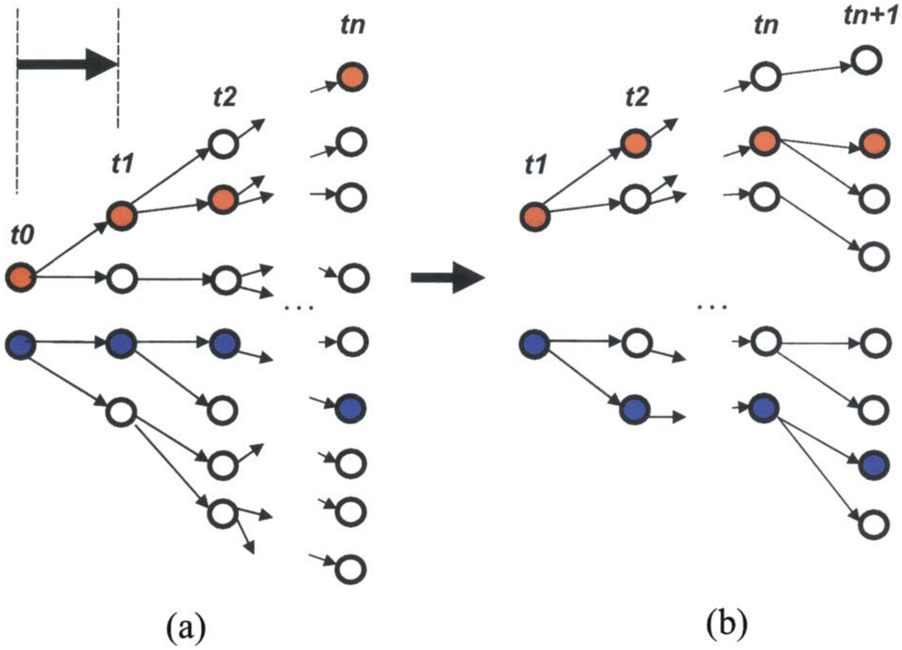

Figure 4-3: N-BFTE example. Each node represents a PHCA state (marking). A trajectory is represented as a sequence of filled nodes. The dotted lines represent a single-step shift of the time horizon.

In addition to incorporating new constraints for observations and commands, as the time horizon is shifted, the T=0 *Constraints* (Section 3.1.5) are removed (part 1.4 of Figure 4-1). Similar to observations and commands, these constraints can be deleted by setting their respective tuples to all possible combinations of assignments.

In order to track the trajectories found from the previous iteration of N-BFTE, a new *Initial State Constraint* (Section 3.1.5) is added. This constraint restricts the start states at Time $t+1$ to match the states of the enumerated trajectories within the previous time horizon. This can be seen in Figure 4-3(b). As the time horizon is shifted from $[t_0..t_n]$ to $[t_1..t_{n+1}]$, the states at t_1 coincide. This means that the initial states of the trajectories within the new horizon are fixed; however, the rest of the trajectory is recomputed within the new horizon. The new trajectories will thus effectively track the prior, while allowing the flexibility to regenerate the rest. This accounts for delayed symptoms that may otherwise result in incorrect diagnoses, as shown in Chapter 2 (Figure 2-6).

The *Initial State Constraint* is formulated in part 1.5 of Figure 4-1. The scope of this constraint is the initial location variables L^t_Σ, given a time horizon $[t..t+N]$. Since the scope of the constraint is fixed, the structure of the constraint can be incorporated into the offline tree decomposition phase. Therefore, the online phase need only generate the tuples and valuations of this constraint. As shown in Figure 4-1, each tuple of the constraint is mapped to a value given by function F_I. The value of each initial state S in the new horizon is obtained by multiplying the probability of the previous state S' from which S stems, by the transition probability from S' to S. These values can be normalized across the number K of maintained trajectories.

Finally, the updated COP is solved using a decomposition-based optimal constraint solver, such as [48, 51, 47]. The complexity of these solvers are bound by structural properties of the tree decomposition. Time complexity is exponential in the maximum number of variables in a tree node, and space complexity is exponential in the number of variables shared between two nodes [31]. This is in contrast to an exponential complexity in the total number of COP variables.

Consider the vision-based navigation scenario in Figure 2-6. If a time horizon $[0..6]$ is used, trajectories will be generated starting from the $(Nominal)$ state at Time 0. Even though the number of trajectories is limited, the trajectory ending at state $(Sensor = Broken)$ at time 6 will have the highest probability based on the delayed observation. Consequently, the state $(Sensor = Broken)$ at Time 2 will be maintained because it is part of the most likely trajectory within the horizon $[0..6]$. As the horizon is shifted to $[1..7]$, the initial constraints IC for probabilistic marking of the start states are removed from the COP (part 1.4 of Figure 4-1). Instead, the trajectories from the previous horizon are tracked by adding a constraint that limits the states at time 1 to correspond to those of the trajectories obtained within the previous horizon (part 1.5 of Figure 4-1). The new constraint for initial states is a valued constraint that maps each initial state to its probability value. The probability of each initial state within the new horizon $[1..7]$ is computed as the probability of its previous state, multiplied by the transition probability to the new initial state, as given in Figure 4-1.

4.3 Parameter Selection

The two parameters of the diagnostic system presented above are the size N of the time horizon, and the number K of maintained trajectories within each horizon. The tradeoff between N and K is as follows. Increasing the number K of enumerated trajectories solves the delayed-symptom problem by maintaining a larger number of trajectories at each time step. In this case, N can be chosen to be small. On the other hand, if a large time horizon is considered, even a single trajectory will be enough to perform diagnosis in the presence of delayed symptoms.

The choice of parameters N and K is typically dependent on the properties of the modeled domain and scenarios. For a system with many combinations of similar failure states with high probability, the number of trajectories maintained will have to be very large in order to be able to account for a delayed symptom that supports an initially low probability state. For such systems, considering even a small number of previous time steps gives enough flexibility to regenerate the correct diagnosis.

4.4 Summary

This chapter introduced the online monitoring and diagnostic process, called N-Stage Best-First Trajectory Enumeration (N-BFTE). This process enumerates and tracks the K most likely trajectories of system state, within an N-Stage time horizon, given observations and issued commands within the horizon. This is accomplished by dynamically updating the tree-decomposed constraint optimization problem (COP), which was generated in the offline phase. More specifically, the COP is updated by shifting the time horizon, incorporating new observations and commands, and inserting constraints for tracking the trajectories found within the previous horizon. Within each time horizon, the COP is solved using an efficient, decomposition-based optimal constraint solver [47]. The next chapter presents the empirical validation of this system on representative scenarios.

Chapter 5

Results and Discussion

The monitoring and diagnosis system presented in the previous chapters has been implemented and empirically validated. This chapter presents the results for representative scenarios, including models of the MIT SPHERES satellites and the NASA Earth Observing One (EO-1) spacecraft.

5.1 Implementation

The N-Stage Best-First Trajectory Enumeration (N-BFTE) capability, described in the previous chapters, has been implemented in C++. Figure 1-2 shows both phases of the N-BFTE process.

In the offline phase, the N-Stage soft COP is generated automatically, given a PHCA model and parameter N. The implementation also supports the use of PCCA models (see Appendix B), used within Livingstone-1 [54], Livingstone-2 [33], and Titan [55] diagnostic engines.

To enhance the efficiency of the solution phase, tree decomposition techniques [24, 31] are applied to decompose the COP into independent subproblems, by exploiting the structural properties of the constraints. Tree decomposition of the COP during the offline compilation phase enables backtrack-free solution extraction during the online phase [17]. The implementation supports the use of different decomposition packages, in particular hypertree decomposition [24], bucket elimination [19], and TOOLBAR

[34] which provides several decomposition schemes. Note that using TOOLBAR in the context of this work requires mapping the VCSP to a Weighted CSP [34].

The online monitoring and diagnosis process uses both the COP formulation and its corresponding tree decomposition, as shown in Figure 1-2. The online phase consists of a loop that implements the pseudocode in Figure 4-1 of Chapter 4. At each iteration of the loop, the updated COP is solved using an implementation of the decomposition-based constraint optimization algorithm in [47].

The following sections present the demonstration scenarios and empirical validation results.

5.2 Demonstration Scenarios

The N-BFTE capability has been validated on several scenarios, which are discussed in this section. First, consider the vision-based navigation scenario introduced in Chapter 1. The diagnoses generated for this scenario were presented in Chapter 2. Table 5.1 shows the size of the COP in terms of the number of variables and constraints generated in the offline phase. As expected, the size of the COP is linear in the size of the time horizon N. Table 5.1 also shows the breakdown of the variables into the three different categories: L_Σ for location (solution) variables, Δ for PHCA variables, and E for auxiliary variables introduced for capturing the PHCA execution semantics (see Section 3.1.2). The distribution of variables across the three categories indicates that E contains the majority of the variables.

For a time horizon with $N = 6$, the COP has 436 variables and 441 constraints, and is solved online in $\sim$ 1.5 sec.

The constraints that simulate the PHCA semantics have also been empirically validated for a number of random PHCA models, such as the examples presented in Chapter 3.

The diagnostic system has further been validated on models of two spacecraft: the MIT SPHERES testbed, and the NASA Earth Observing One spacecraft.

N	$Constraints$	$Variables$	L_{Σ}	Δ	E
1	91	96	8	22	66
2	161	164	2	33	119
3	231	232	16	44	172
4	301	300	20	55	225
5	371	368	24	66	278
6	441	436	28	77	331
7	511	504	32	88	384
8	581	578	36	99	437
9	651	640	40	110	490
10	721	708	44	121	543

Table 5.1: Number of constraints and variables generated for the camera model in Figure 2-5. N is the size of the time horizon.

5.2.1 SPHERES Global Metrology

SPHERES (Synchronized Position Hold Engage and Reorient Experimental Satellites) [45] is a formation flying upgradeable testbed, developed at the Space Systems Laboratory at MIT, for demonstrating novel metrology, control and autonomy technologies on the International Space Station (Figure 5-1).

Two representative models [41] of the SPHERES Global Metrology subsystem were used for demonstrating the capability in this work. The global metrology subsystem provides measurements to update estimates of position and attitude for each SPHERES satellite. This is accomplished by using ultrasonic time of flight measurements from five external beacons to ultrasound sensors on the SPHERES surfaces to

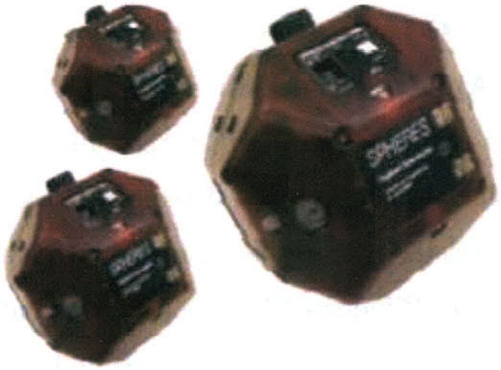

Figure 5-1: The SPHERES testbed. [32, 45]

determine attitude and position with respect to a global reference frame.

The following is an example of a complex interaction that may occur on SPHERES: The thrusters on each SPHERES satellite generate significant ultrasonic noise when in use, which interferes with the global measurement system. Therefore, the thrusters must be turned off during global measurements. Otherwise, the signal from the beacons will be corrupted and the global measurement will be unreliable.

Modeled SPHERES components include beacons, sensors, ultrasound signals, global measurements, and thrusters that may interfere with the ultrasound signals, possibly causing failures. These models are described in [41].

The first model (SPHERES 1) consists of 5 components, and the second model (SPHERES 2) consists of 18 components. The results are presented in Section 5.3.

5.2.2 Earth Observing One

Earth Observing One (EO-1) is a NASA New Millennium spacecraft which has validated a number of instrument and spacecraft technologies. Models of the EO-1 spacecraft were developed at NASA and used for validating L2 on EO-1 [28]. The EO-1 model consists of 12 components, which are parts of the Advanced Land Imager, the Hyperion instrument and the Wideband Advanced Recorder Processor onboard the EO-1.

5.3 Results

Both the SPHERES and EO-1 systems are modeled as probabilistic, concurrent constraint automata (PCCA) [54]. The definition of a PCCA is included in Appendix A. A PCCA model is a special case of PHCA that allows for parallel composition of probabilistic constraint automata, but not sequential composition. Nevertheless, these models provided realistic scenarios for validating the approach in this work. All experiments were run under Windows XP on a 1.6 GHz Pentium M processor.

Figure 5-2 shows the size of the COP generated offline, for each of the models. Recall that SPHERES 1 consists of 5 components, SPHERES 2 consists of 18 compo-

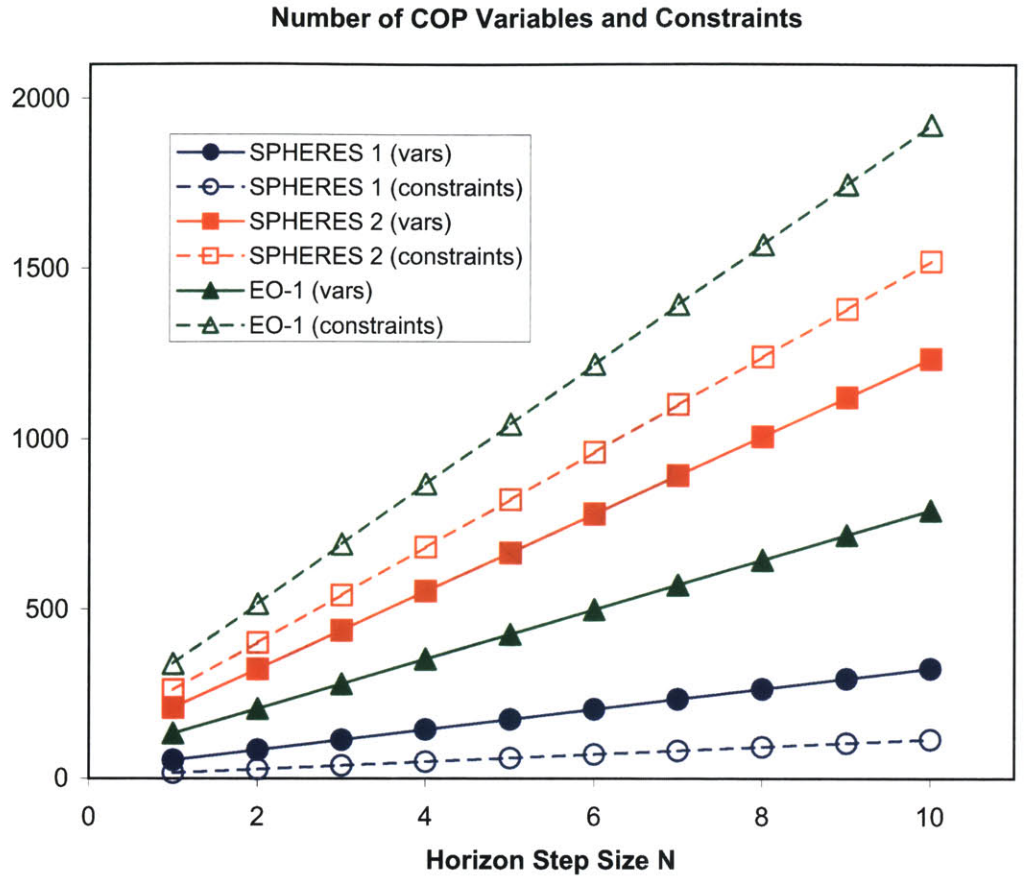

Figure 5-2: Size of the COP generated offline for the SPHERES and EO-1 models.

nents, and EO-1 consists of 12 components. The EO-1 models took the longest time to compile to a soft COP. This is because the domain sizes for the EO-1 models are on average larger than those of the SPHERES models.

For each of these models, the COP generated in the offline phase was decomposed using bucket elimination. For instance, Figure 5-3 shows the constraint graph of the EO-1 model, for $N = 1$. This graph was generated using the TOOLBAR package [51]. The nodes of the constraint graph represent the variables of the COP, similar to a hypergraph. The edges of the graph connect variables that appear in the same hyperedge of a hypergraph [17]. Figure 5-4 shows the decomposition of the constraint graph into a tree structure, also generated using TOOLBAR. This tree-structured COP is solved online, generating a single trajectory in 0.16 seconds.

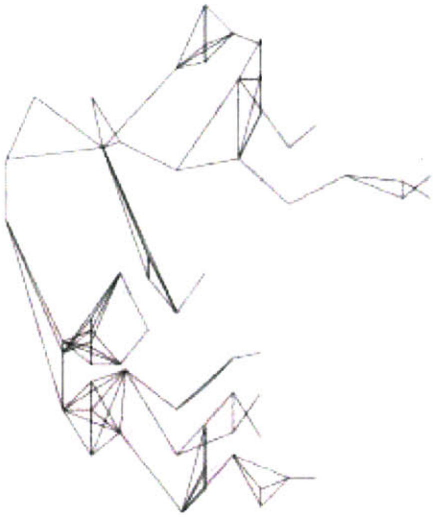

Figure 5-3: Constraint graph of the EO-1 model for $N = 1$.

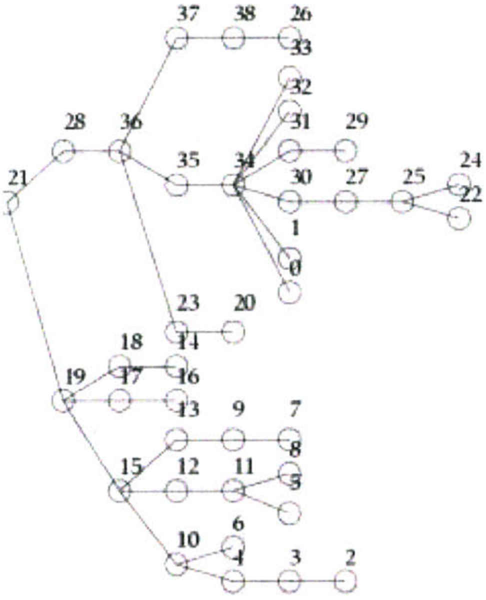

Figure 5-4: Tree decomposition of the EO-1 constraint graph in Figure 5-3

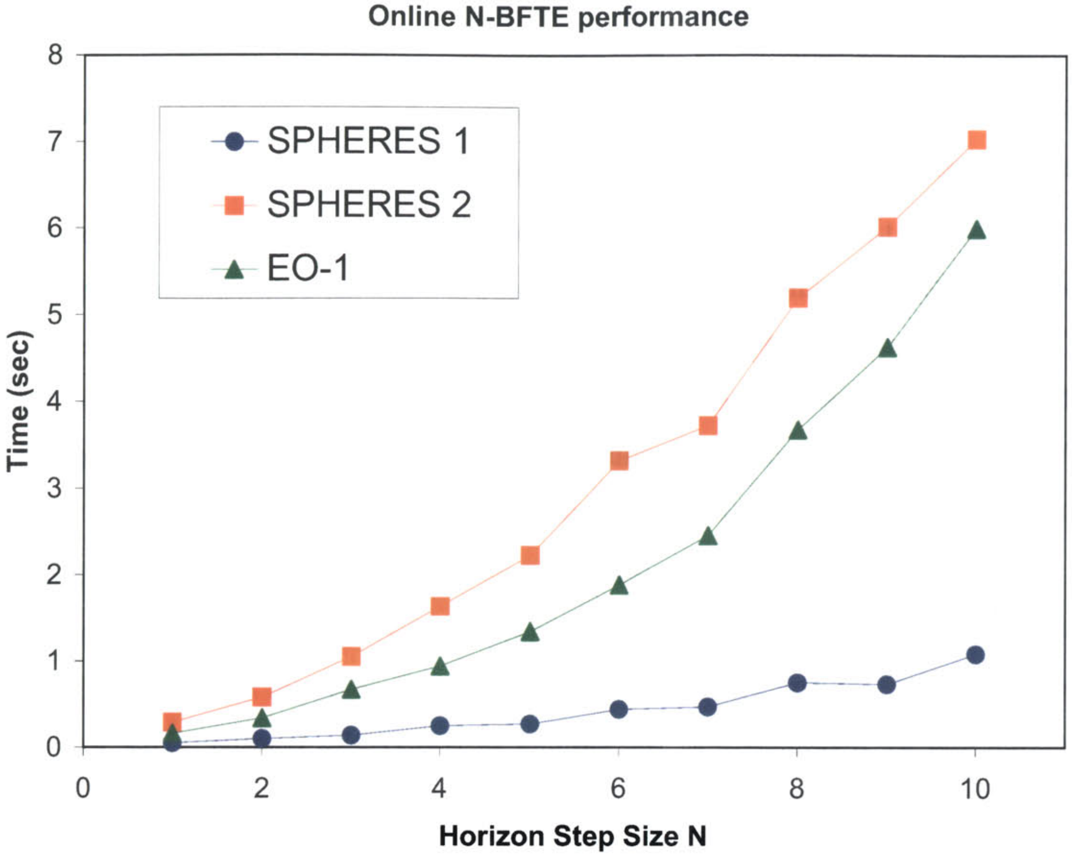

Figure 5-5: Online performance for the SPHERES and EO-1 models.

Figure 5-5 shows the time required for solving the online N-BFTE problem. The results for the online phase were obtained for various nominal and failure scenarios as a function of the time horizon size N, for the same number of trajectories tracked (K=1 in Figure 5-5). For each of the tested scenarios, N-BFTE successfully generated the correct diagnosis based on delayed observations within the time horizon N. For instance, in the SPHERES scenarios most symptoms exhibited delays of 3 or 4 time steps. Choosing a time horizon of $N = 4$ was enough to account for all delayed symptoms.

The implementation and empirical validation results on the representative scenarios above are a proof of concept of the approach presented in this work. More complex PHCA models are currently being developed for an adaptive vision system [46], to navigate a network of robotic explorers.

5.4 Future Work

Future work relates to enhancing the efficiency of diagnosis, handling delayed symptoms effectively, and the extension of this work to other applications.

Enhancing the efficiency of diagnosis can be addressed by unifying decomposition-based [24, 10] and conflict-directed [55] techniques into the optimal constraint solver. This unification would leverage the advantages of two major approaches to achieving efficiency in diagnostic systems: exploiting system structure and exploiting conflicts. Future work also includes optimizing the offline compilation step, to minimize the number of variables and constraints generated.

With respect to diagnosis in the presence of delayed symptoms, the parameters N and K of the diagnostic system are typically dependent on a given scenario and domain model. Future research may investigate the optimal size of the diagnosis horizon and its relationship to the number of trajectories tracked. This will allow for automated parameter selection for the diagnostic system, based on properties of the modeled domain. Accuracy of the diagnostic capability can also be addressed through observation and transition guard probabilities. This work used an upper bound approximation to observation probabilities and transition guard probabilities, as described in Chapter 3. The diagnostic system can be improved by using better estimates of the observation probability and transition guard probability function in Equation 2.1.

In addition to monitoring and diagnosis of complex hardware and software systems, this work has several new applications, including distributed diagnosis, verification and planning [44]. Traditionally, model-based verification of embedded systems has focused on determining program correctness using techniques such as symbolic model checking [29]. However, verification is performed offline during design and development, and is not guaranteed to verify against all possible system failures. This work applies to probabilistic online monitoring from PHCA specifications, as well as to verification of embedded software, given fault models. Another application is to leverage structural tree decomposition to develop a distributed variant of this work

[8], by mapping individual tree nodes to different processors. Finally, this work can be used in the context of finite-horizon planning and execution, similar to [1]. In particular, extending the system to PHCA-based reconfiguration will result in a unified fault diagnosis and recovery capability that can support complex, software-extended models.

5.5 Conclusions

This thesis introduced a capability for model-based diagnosis of systems with complex hardware and software behavior. The key contributions of this capability include:

- Diagnosis of complex systems, based on an expressive modeling formalism (PHCA), and its encoding as a soft constraint optimization problem.
- Complex systems diagnosis in the presence of delayed symptoms, through N-Stage best first trajectory enumeration (N-BFTE).
- Efficient online diagnosis, by leveraging state-of-the-art constraint decomposition techniques.
- Validation against representative spacecraft and robotic domains.

Appendix A

List of PHCA Encoding Constraints

The following is a summary of the constraints presented in Chapter 3, for encoding Probabilistic Hierarchical Constraint Automata (PHCA):

State Constraint Consistency:

$$\forall t \in \{0..N\}, \forall L \in \Sigma : State_L^t = Consistent \Leftrightarrow C(L)^t \tag{A.1}$$

Transition Guard Consistency:

$$\forall t \in \{0..N-1\}, \forall \tau \in T : Guard_\tau^t = Consistent \Leftrightarrow C(\tau)^t \tag{A.2}$$

T=0 Model Marking:

$$\forall C \in (PHCAModels) : C^0 = Marked \tag{A.3}$$

T=0 Unmarking:

$$\forall C \in \Sigma_C, \forall L \in (Subautomata(C) - \Theta(C)) : L^0 = Unmarked \tag{A.4}$$

T=0 Full Marking:

$$\forall C \in \Sigma_C : [C^0 = Marked \Leftrightarrow (\forall L \in \Theta(C) : Start_L^0 = Enabled)]$$
$$\wedge [C^0 = Unmarked \Leftrightarrow (\forall L \in \Theta(C) : Start_L^0 = Disabled)] \quad (A.5)$$

T=0 Probabilistic Marking:

$$\forall C \in \Sigma_C, \forall L \in \Theta(C) : L^0 = Marked \Rightarrow$$
$$(Start_L^0 = Enabled \wedge State_L^0 = Consistent) \quad (A.6)$$

with valuation function

$$F_0(M) = \begin{cases} Prob(L^0 = Marked) & if\ Condition1 \\ 1 - Prob(L^0 = Marked) & if\ Condition2 \\ 1.0 & otherwise \end{cases} \quad (A.7)$$

where $Condition1$ is:

$$(L^0 = Marked) \wedge (Start_L^0 = Enabled) \wedge (State_L^0 = Consistent) \quad (A.8)$$

and $Condition2$ is:

$$(L^0 = Unmarked) \wedge (Start_L^0 = Enabled) \wedge (State_L^0 = Consistent) \quad (A.9)$$

Probabilistic Transition Choice:

$$\forall t \in \{0..N-1\}, \forall P \in \Sigma_P : (\exists \tau \in T : Source(\tau) = P \Rightarrow$$
$$[P^t = Marked \Leftrightarrow (\exists T \in \{T | Source(T) = P\} : T^t = Enabled$$
$$\wedge (\forall T' \in (\{T | Source(T) = P\} - \{T\}) : T'^t = Disabled))] \bigwedge$$
$$[P^t = Unmarked \Leftrightarrow (\forall T \in \{T | Source(T) = P\} : T^t = Disabled)]) \quad (A.10)$$

with valuation function:

$$F_T(M) = \begin{cases} Prob(T_i) & if\ (\exists T_i^t : T_i^t = Enabled) \\ 1.0 & otherwise \end{cases} \tag{A.11}$$

Transition Consistency:

$$\begin{aligned} &\forall t \in \{0..N-1\}, \forall \tau \in T : \tau^t = Enabled \Rightarrow \\ &(Source(\tau)^t = Marked \wedge Guard_\tau^t = Consistent) \end{aligned} \tag{A.12}$$

Target Identification:

$$\begin{aligned} &\forall t \in \{0..N-1\}, \forall L \in \Sigma : TransitionTo_L^{(t+1)} = Enabled \Leftrightarrow \\ &(\exists \tau \in \{T | Target(T) = L\} : \tau^t = Enabled) \end{aligned} \tag{A.13}$$

Target Full Marking:

$$\begin{aligned} &\forall t \in \{1..N\}, \forall C \in \Sigma_C : \\ &[(\forall L \in \Theta(C) : Start_L^t = Enabled) \Leftrightarrow \\ &(TransitionTo_C^t = Enabled \vee Start_C^t = Enabled)] \bigwedge \\ &[(\forall L \in \Theta(C) : Start_L^t = Disabled) \Leftrightarrow \\ &(TransitionTo_C^t = Disabled \wedge Start_C^t = Disabled)] \end{aligned} \tag{A.14}$$

Primitive Target Marking:

$$\begin{aligned} &\forall t \in \{1..N\}, \forall P \in \Sigma_P : [P^t = Marked \Leftrightarrow \\ &(TransitionTo_P^t = Enabled \vee Start_P^t = Enabled) \wedge State_P^t = Consistent] \\ &\bigwedge [P^t = Unmarked \Leftrightarrow (TransitionTo_P^t = Disabled \wedge Start_P^t = Disabled)] \end{aligned}$$

Composite Target Marking:

$$\begin{aligned}
&\forall t \in \{1..N\}, \forall C \in \Sigma_C : [C^t = Marked \Rightarrow State^t_C = Consistent] \\
&\bigwedge [C^t = Marked \Leftarrow (TransitionTo^t_C = Enabled \vee Start^t_C = Enabled) \\
&\wedge State^t_C = Consistent] \\
&\bigwedge [C^t = Unmarked \Rightarrow (TransitionTo^t_C = Disabled \\
&\wedge Start^t_C = Disabled)]
\end{aligned} \tag{A.15}$$

Hierarchical Composite Marking/Unmarking:

$$\begin{aligned}
&\forall t \in \{0..N\}, \forall C \in \Sigma_C : \\
&[C^t = Marked \Leftrightarrow (\exists L \in Subautomata(C) : L^t = Marked)] \bigwedge \\
&[C^t = Unmarked \Leftrightarrow (\forall L \in Subautomata(C) : L^t = Unmarked)]
\end{aligned} \tag{A.16}$$

Appendix B

Probabilistic Concurrent Constraint Automata

A probabilistic constraint automaton for component "a" is defined by the tuple $\mathcal{A}_a = \langle \Pi_a, \mathbb{M}_a, \mathbb{T}_a, \mathbf{P}_{\mathbb{T}_a} \rangle$:

1. $\Pi_a = \Pi_a^m \cup \Pi_a^r$ is a finite set of discrete variables for component "a", where each variable $\pi_a \in \Pi_a$ ranges over a finite domain $\mathbb{D}(\pi_a)$. Π_a^m is a singleton set containing *mode* variable $\{x_a\} = \Pi_a^m$ whose domain $\mathbb{D}(x_a)$ is the finite set of discrete modes in $\mathcal{A}_a$. *Attribute* variables Π_a^r include inputs, outputs, and any other variables used to define the behavior of the component. Σ_a is the complete set of all possible full assignments over Π_a and the state space of the component $\Sigma_a^{x_a} = \Sigma_a \Downarrow_{x_a}$ is the projection of Σ_a onto mode variable x_a.

2. $\mathbb{M}_a : \Sigma_a^{x_a} \rightarrow \mathbb{C}(\Pi_a^r)$ maps each mode assignment $(x_a = v_a) \in \Sigma_a^{x_a}$ to a finite domain constraint $c_a(x_a = v_a) \in \mathbb{C}(\Pi_a^r)$, where $\mathbb{C}(\Pi_a^r)$ is the set of finite domain constraints over Π_a^r. These constraints are known as *modal constraints* and are typically encoded in the propositional form $\lambda \triangleq \textbf{True} \mid \textbf{False} \mid (u = y) \mid \neg\lambda_1 \mid \lambda_1 \wedge \lambda_2 \mid \lambda_1 \vee \lambda_2$, where $y \in \mathbb{D}(u)$. If the current mode is $(x_a^t = v_a)$ at time-step t, then the assignments to each attribute variable $r_a^t \in \Pi_a^r$ at time-step t must be consistent with $c_a(x_a = v_a)$. These constraints capture the physical behavior of the mode.

3. $\mathbb{T}_a : \Sigma_a^{x_a} \times \mathbb{C}(\Pi_a^r) \rightarrow \Sigma_a^{x_a}$ is a set of transition functions. The set of finite domain constraints $\mathbb{C}(\Pi_a^r)$ are also known as the *transition guards*, encoded in the propositional form λ. Given a current mode assignment $(x_a = v_a) \in \Sigma_a^{x_a}$ and guard $g_a \in \mathbb{C}(\Pi_a^r)$ entailed at time-step t, each transition function $\tau_a(x_a = v_a, g_a) \in \mathbb{T}_a(x_a = v_a, g_a)$ specifies a target mode assignment $(x_a = v_a') \in \Sigma_a^{x_a}$ that the automaton could transition into at time-step $t + 1$. $\mathbb{T}_a = \mathbb{T}_a^n \cup \mathbb{T}_a^f$ captures both nominal and faulty behavior.

4. $\mathbf{P}_{\mathbb{T}_a} : \mathbb{T}_a(x_a = v_a, g_a) \rightarrow \Re[0, 1]$ is a transition probability distribution. For each mode variable assignment in $\Sigma_a^{x_a}$ and guard g_a^t, there is a probability distribution across all transitions into target modes defined by the set of transition functions $\mathbb{T}_a(x_a = v_a, g_a)$.

The PCCA plant model is then defined by the tuple $\mathcal{P} = \langle \mathcal{A}, \Pi, \mathbb{Q} \rangle$:

1. $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n\}$ is the finite set of constraint automata that represent the n components of the plant.

2. $\Pi = \bigcup_{a=1..n} \Pi_a$ is the set of all plant variables. The variables Π are partitioned into a finite set of *mode* variables $\Pi^m = \bigcup_{a=1..n} \Pi_a^m$, *control* variables $\Pi^c \subseteq \bigcup_{a=1..n} \Pi_a^r$, *observation* variables $\Pi^o \subseteq \bigcup_{a=1..n} \Pi_a^r$, and *dependent* variables $\Pi^d \subseteq \bigcup_{a=1..n} \Pi_a^r$. Σ^c, Σ^o, and Σ^d are the sets of full assignments over Π^c, Π^o, and Π^d.

3. $\mathbb{Q} \subset \mathbb{C}(\Pi)$ is a set of finite domain constraints that capture the interconnections between plant components.

Bibliography

[1] A. Barrett. From hybrid systems to universal plans via domain compilation. In *Proc. 14th International Conference on Automated Planning and Scheduling*, 2004.

[2] R. Barták. Modelling soft constraints: a survey. *Neural Network World*, 12(5):421–431, 2002.

[3] L. Baum and T. Petrie. Statistical inference for probabilistic functions of finite-state markov chains. *Annals of Mathematical Statistics*, 37:1554–1563, 1966.

[4] G. Berry and G. Gonthier. The *Esterel* programming language: design, semantics and implementation. *Science of Computer Programming*, 19(2):87–152, Nov. 1992.

[5] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint satisfaction and optimization. *Journal of the ACM*, 44(2):201–236, 1997.

[6] S. Chein, S. T. Debban, C. Yen, R. Sherwood, R. Castano, B. Cichy, A. G. Davies, M. Burl, A. Fukunaga, R. Greeley, T. Doggett, K. Williams, V. Baker, and J. Dohm. Revolutionary deep space science missions enabled by onboard autonomy. In *Proc. 7th International Symposium on Artificial Intelligence, Robotics and Automation in Space*, 2003.

[7] S. Chung, J. Van Eepoel, and B. C. Williams. Improving model-based mode estimation through offline compilation. In *Proc. 6th International Symposium on Artificial Intelligence, Robotics and Automation in Space*, 2001.

[8] S. H. Chung and A. C. Barrett. Distributed real-time model-based diagnosis. In *Proc. IEEE Aerospace Conference*, 2003.

[9] L. Console, G. Friedrich, and D. Theseider Dupre. Model-based diagnosis meets error diagnosis in logic programs. In *Proc. International Joint Conference on Artificial Intelligence*, 1993.

[10] A. Darwiche and G. Provan. Exploiting system structure in model-based diagnosis of discrete-event systems. In *Proc. 7th International Workshop on Principles of Diagnosis*, 1996.

[11] A. Darwiche and G. Provan. Model-based diagnosis using structured system descriptions. *Journal of Artificial Intelligence Research*, 8:165–222, 1998.

[12] J. de Kleer. Using crude probability estimates to guide diagnosis. *Artificial Intelligence*, 45(3):381–391, 1990.

[13] J. de Kleer. Focusing on probable diagnoses. In *National Conference on Artificial Intelligence*, 1991.

[14] J. de Kleer, A. K. Mackworth, and R. Reiter. Characterizing diagnoses and systems. *Artificial Intelligence*, 56(2-3):197–222, 1992.

[15] J. de Kleer and B. C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97–130, 1987.

[16] J. de Kleer and B. C. Williams. Diagnosis with behavioral modes. In *Proc. International Joint Conference on Artificial Intelligence*, 1989.

[17] R. Dechter. *Constraint Processing.* Morgan Kaufmann, 2003.

[18] R. Dechter and A. Dechter. Structure-driven algorithms for truth maintenance. *Artificial Intelligence*, 82:1–20, 1996.

[19] R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38:353–366, 1989.

[20] O. Dressler and P. Struss. Model-based diagnosis with the default-based diagnosis engine: Effective control strategies that work in practice. In *Proc. European Conference on Artificial Intelligence*, 1994.

[21] O. Dressler and P. Struss. The consistency-based approach to automated diagnosis of devices. In *Proc. 5th Principles of Knowledge Representation and Reasoning*, 1996.

[22] H. Fargier and J. Lang. Uncertainty in constraint satisfaction problems: a probabilistic approach. In *Proc. European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, 1993.

[23] P. Fröhlich, W. Nejdl, K. Jobmann, and H. Wietgrefe. Model-based alarm correlation in cellular phone networks. In *Proc. International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, 1997.

[24] G. Gottlob, N. Leone, and F. Scarcello. A comparison of structural CSP decomposition methods. *Artificial Intelligence*, 124(2):243–282, 2000.

[25] I. Grosclaude. Model-based monitoring of component-based software systems. In *Proc. 14th International Workshop on Principles of Diagnosis*, 2003.

[26] W. Hamscher, L. Console, and J. de Kleer. *Readings in Model-Based Diagnosis.* Morgan Kaufmann, 1992.

[27] D. Harel. Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987.

[28] S. Hayden, A. Sweet, and S. Christa. Livingstone model-based diagnosis of earth observing one. In *Proc. 1st AIAA Intelligent Systems*, 2004.

[29] K. L. McMillan D. L. Dill J. R. Burch, E. M. Clarke and J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Information and Computation*, volume 98(2), pages 142–170, 1992.

[30] G. D. Forney Jr. The Viterbi algorithm. *Proceedings of the IEEE*, pages 267–278, 1978.

[31] K. Kask, R. Dechter, and J. Larrosa. Unifying cluster-tree decompositions for automated reasoning. Technical report, U. of California at Irvine, 2003.

[32] E. M. Kong, A. Saenz-Otero, S. Nolet, D. Berkovitz, D. W. Miller, and S. W. Sell. Spheres as a formation flight algorithm development and validation testbed: current progress and beyond. In *Proc. 2nd International Symposium on Formation Flying Missions and Technologies*, 2004.

[33] J. Kurien and P. Nayak. Back to the future for consistency-based trajectory tracking. In *Proc. 17th National Conference on Artificial Intelligence*, 2000.

[34] J. Larrosa and T. Schiex. In the quest of the best form of local consistency for weighted CSP. In *Proc. International Joint Conference on Artificial Intelligence*, 2003.

[35] N. G. Leveson and C. S. Turner. An investigation of the Therac-25 accidents. *IEEE Computer*, 26(7):18–41, 1993.

[36] K. Marriott and P. J. Stuckey. *Programming with Constraints: An Introduction.* MIT Press, 1998.

[37] O. B. Martin. Accurate belief state update for probabilistic constraint automata. Master's thesis, MIT, 2005.

[38] O. B. Martin, B. C. Williams, and M. D. Ingham. Diagnosis as approximate belief state enumeration for probabilistic concurrent constraint automata. In *Proc. 20th National Conference on Artificial Intelligence*, 2005.

[39] C. Mateis, M. Stumptner, and F. Wotawa. Modeling java programs for diagnosis. In *Proc. 14th European Conference on Artificial Intelligence*, 2000.

[40] W. Mayer and M. Stumptner. Approximate modeling for debugging of program loops. In *Proc. 15th International Workshop on Principles of Diagnosis*, 2004.

[41] T. Mikaelian, M. Ouimet, M. Voightmann, and M. Alighanbari. Modeling the spheres global metrology system, project report, 2003.

[42] T. Mikaelian, B. C. Williams, and M. Sachenbacher. Diagnosing complex systems with software-extended behavior using constraint optimization. In *Proc. 16th International Workshop on Principles of Diagnosis*, 2005.

[43] T. Mikaelian, B. C. Williams, and M. Sachenbacher. Model-based monitoring and diagnosis of systems with software-extended behavior. In *Proc. 20th National Conference on Artificial Intelligence*, 2005.

[44] T. Mikaelian, B. C. Williams, and M. Sachenbacher. Probabilistic monitoring from mixed hardware and software specifications. In *Proc. International Conference on Automated Planning and Scheduling, Workshop on Verification and Validation of Model-based Planning and Scheduling Systems*, 2005.

[45] S. Nolet, E. Kong, and D. W. Miller. Autonomous docking algorithm development and experimentation using the spheres testbed. In *Proc. SPIE Defense and Security Symposium*, 2004.

[46] P. Robertson and J. M. Brady. Adaptive image analysis for aerial surveillance. *IEEE Intelligent Systems*, pages 30–45, 1999.

[47] M. Sachenbacher and B. C. Williams. Diagnosis as semiring-based constraint optimization. In *Proc. European Conference on Artificial Intelligence*, 2004.

[48] M. Sachenbacher and B. C. Williams. On-demand bound computation for best-first constraint optimization. In *Proc. International Conference on Principles and Practice of Constraint Programming*, 2004.

[49] M. Sachenbacher and B. C. Williams. Diagnosis using bounded search and symbolic inference. In *Proc. International Workshop on Principles of Diagnosis*, 2005.

[50] T. Schiex, H. Fargier, and G. Verfaillie. Valued constraint satisfaction problems:hard and easy problems. In *Proc. International Joint Conference on Artificial Intelligence*, 1995.

[51] C. Terrioux and Philippe Jégou. Bounded backtracking for the valued constraint satisfaction problems. In *Proc. 9th International Conference on Principles and Practice of Constraint Programming*, 2003.

[52] B. C. Williams, S. Chung, and V. Gupta. Mode estimation of model-based programs: monitoring systems with complex behavior. In *Proc. 17th International Joint Conference on Artificial Intelligence*, 2001.

[53] B. C. Williams, M. Ingham, S. H. Chung, and P. H. Elliott. Model-based programming of intelligent embedded systems and robotic space explorers. *Proceedings of the IEEE: Special Issue on Modeling and Design of Embedded Software*, 9(1):212–237, 2003.

[54] B. C. Williams and P. Nayak. A model-based approach to reactive self-configuring systems. In *Proc. 13th National Conference on Artificial Intelligence*, pages 971–978, 1996.

[55] B. C. Williams and R. Ragno. Conflict-directed A* and its role in model-based embedded systems. *To appear, Journal of Discrete Applied Math (accepted 2001).*